RANDOMISIERTE ALGORITHMEN

WS 2004/05

Version v. 20.08.2006

Dozent:

Thomas Hofmeister

Universität Dortmund

Lehrstuhl Informatik 2

44221 Dortmund

Vorwort

Dieses Skript ist aus den Mitschriften entstanden, die die Hörer/-innen meiner Vorlesung dankenswerterweise erstellt haben. Ich habe die Mitschriften jeweils korrekturgelesen und Änderungen daran direkt eingearbeitet.
Für die Rechtschreibung wurde in diesem Skript ein randomisiertes Verfahren verwendet: Mit Wahrscheinlichkeit p_{alt} wurden Worte nach der alten Rechtschreibung korrekt geschrieben, mit Wahrscheinlichkeit p_{neu} nach der neuen und für ein (hoffentlich) kleines $\varepsilon > 0$ nach keiner von beiden.
Über konkrete Hinweise auf Fehler oder Verbesserungsmöglichkeiten freue ich mich immer.
Meine email-Adresse findet sich auf meiner Homepage:
`http://ls2-www.cs.uni-dortmund.de/~hofmeister/`.
Insbesondere danke ich:

Klaus Fehlker, Patrick Flecken, Dominik Göddeke, Steffen Gregorius, Bastian Krol, Nicole Skaradzinski, Tobias Storch und Dirk Sudholt.
Außerdem bedanke ich mich bei Thomas Franke für die Hilfe bei der Überarbeitung des Skriptes in der Version WS 2004/05.

Was wurde in der Vorlesung im WS 2004/05 geschafft?

Alles außer:
– das Kapitel über Online-Algorithmen
– das Kapitel über den Approximationsalgorithmus mit Güte 0.878 für MAXCUT

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Vorlesung wird sowohl grundlegende randomisierte Algorithmen für verschiedene Bereiche vorstellen, als auch Methoden und Werkzeuge zur Analyse von randomisierten Algorithmen vermitteln.
Literatur: Motwani/Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

1.1 Vorteile randomisierter Algorithmen

Prinzipiell bieten randomisierte Algorithmen sowohl Vorteile hinsichtlich der Effizienz — sie haben im Vergleich zu deterministischen Algorithmen häufig eine bessere Laufzeit und einen besseren Speicherplatzbedarf — als auch hinsichtlich der Einfachheit — sie sind oft einfacher zu implementieren.

Prinzipien (nach Motwani/Raghavan)

- *foiling an adversary* (dt. Vereitelung der Angriffe eines Gegners):
  Das Problem wird als Spiel zwischen dem Algorithmendesigner und einem natürlichen Gegner (die Schwierigkeit des Problems) aufgefasst. Wegen der Randomisierung kann der Gegner nicht alle Schritte kennen und hat daher Probleme, eine *worst-case-Eingabe* zu konstruieren.

- *random sampling* (dt. Zufallsstichproben):
  Eine kleine Menge eines großen Suchraums kann stellvertetend ähnliche Eigenschaften wie die gesamte Menge haben.

- *abundance of witnesses* (dt. Überfluss an Zeugen)

- *fingerprinting and hashing* (dt. Signaturen und Fingerabdrücke):
  Große Objekte werden durch kleine Objekte dargestellt. Z.B. wird bei einem Vergleich von großen Dateien erst die Dateigröße verglichen oder allgemeiner die Werte einer Hashfunktion, die auf die Dateien angewendet wurde. Dieses Vorgehen spielt als CRC-Check bei der Datenübertragung eine wichtige Rolle.

- *random reordering* (dt. zufällige Umordnung):
  Mit hoher Wahrscheinlichkeit werden *worst-case-Eingaben* ausgeschlossen.

- *load balancing* (dt. Lastbalancierung):
  Man versucht die auftretende Last gleichmäßig, z.B. auf mehrere Prozessoren, zu verteilen.

- *rapidly mixing Markov chains* (dt. schnell mischende Markovketten):
  Ein Markovprozess wird zur Erzeugung einer guten Stichprobe genutzt.

- *isolation and symmetry breaking* (dt. Isolierung und Aufbrechen von Symmetrien):
  Dieses Prinzip stellt ein gutes Mittel bei parallelen Algorithmen dar.
- *probabilistic methods and existence proofs* (dt. probabilistische Methoden und Existenzbeweise):
  Ziel ist der Nachweis, dass ein Algorithmus ein Objekt mit bestimmten Eigenschaften mit Wahrscheinlichkeit größer 0 berechnet.

Randbemerkung: Manche Autoren bevorzugen den Ausdruck „probabilistische Algorithmen“ gegenüber dem Ausdruck „randomisierte Algorithmen“.

1.2 Einfache Begriffe aus der Wahrscheinlichkeitstheorie

Mathematische Formulierung für den Ausgang eines Zufallsversuchs ist die Zufallsvariable (ZV). Für uns reicht folgende Definition:

1.1 Definition (ZV)**.** Eine Zufallsvariable X ist gegeben durch eine Trägermenge M_X und durch eine Abbildung

$$p : M_X \to [0,1] \text{ mit } \sum_{m \in M_X} p(m) = 1.$$

1.2 Definition (Erwartungswert)**.** Für eine Zufallsvariable X, deren Trägermenge M_X eine Teilmenge der reellen Zahlen ist, kann man den Erwartungswert definieren:

$$\mathbf{E}[X] := \sum_{m \in M_X} p(m) \cdot m.$$

1.3 Beispiel. $\mathbf{E}[\text{Würfel}] = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \cdots + \frac{1}{6} \cdot 6 = 3.5.$

Bei unendlichen Mengen muss man natürlich darauf achten, ob die unendliche Summe konvergiert oder nicht. Wir werden typischerweise mit höchstens abzählbar unendlichen Trägermengen zu tun haben.

Bemerkung: Wenn eine Zufallsvariable X die Trägermenge $M_X = \{0,1\}$ hat, dann gilt $\mathbf{E}[X] = \mathbf{Prob}(X = 1) \cdot 1 + \mathbf{Prob}(X = 0) \cdot 0 = \mathbf{Prob}(X = 1)$.

Linearität des Erwartungswertes: Erwartungswerte verhalten sich bei Addition linear. Seien X_1 und X_2 Zufallsvariablen, dann ist

$$\mathbf{E}[X_1 + X_2] = \mathbf{E}[X_1] + \mathbf{E}[X_2].$$

Dies gilt *auch für nicht unabhängige* Zufallsvariable. Im Vergleich dazu bei der Multiplikation:

$$\mathbf{E}[X_1 \cdot X_2] = \mathbf{E}[X_1] \cdot \mathbf{E}[X_2] \text{ gilt } \textit{nur für unabhängige} \text{ Zufallsvariable.}$$

1.4 Beispiel. 40 Matrosen kehren in ihre 40 Kajüten zurück, jeder wählt gemäß Gleichverteilung eine Kajüte. Wie viele erwarten wir in der richtigen, d.h. ihrer eigenen Kajüte? Um dieses zu berechnen, definieren wir so genannte „Indikatorvariablen“, die nur die Werte 0 oder 1 annehmen können:

$$Y_i := \begin{cases} 1, & \text{falls Matrose } i \text{ in (richtiger) Kajüte } i. \\ 0 & \text{sonst.} \end{cases}$$

Für uns interessant:

$$\mathbf{E}[Y_1 + \cdots + Y_n] = \mathbf{E}[Y_1] + \cdots + \mathbf{E}[Y_n] = 40 \cdot \mathbf{E}[Y_1] = 40 \cdot [\frac{1}{40} \cdot 1 + \frac{39}{40} \cdot 0] = 1.$$

Eine kompliziertere Rechnung über die Definition des Erwartungswertes hätte vielleicht wie folgt ausgesehen:

$$\sum_{i=0}^{40} \mathbf{Prob}\,(\text{genau } i \text{ gehen in richtige Kajüte}) \cdot i = \sum_{i=0}^{40} \left(\begin{array}{c} 40 \\ i \end{array} \right) \cdot \left(\frac{1}{40}\right)^{i} \cdot \left(\frac{39}{40}\right)^{40-i} \cdot i = \ldots ?$$

1.5 Beispiel. Angenommen, man hat ein Ereignis, das mit Wahrscheinlichkeit p eintritt. Wie groß ist die Wahrscheinlichkeit, dass bei n durchgeführten Versuchen das Ereignis ungerade oft eintritt? Die Definition des Erwartungswertes führt zu folgendem Ansatz:

$$\sum_{\substack{i=1,\\ i \text{ ungerade}}}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) \cdot p^i \cdot (1-p)^{n-i} = \ldots ?$$

Über die Linearität des Erwartungswertes gelangt man eleganter zu folgender allgemeineren Aussage:

1.6 Satz. *Gegeben seien n unabhängige Zufallsvariablen $y_1, y_2, \ldots, y_n$ mit* $\mathbf{Prob}\,(y_i = -1) = p_i$ *und* $\mathbf{Prob}\,(y_i = 1) = 1 - p_i$. *Dann gilt:*

$$\mathbf{Prob}\,(y_1 \cdot y_2 \cdots y_n = -1) = \frac{1 - \prod_{i=1}^{n}(1-2p_i)}{2}.$$

Beweis: Es ist $\mathbf{E}\,[y_i] = p_i \cdot (-1) + (1-p_i) \cdot (+1) = 1 - 2p_i$. Wegen der Unabhängigkeit der y_i folgt:

$$\mathbf{E}\,[y_1 \cdot y_2 \cdots y_n] = \mathbf{E}\,[y_1] \cdot \mathbf{E}\,[y_2] \cdots \mathbf{E}\,[y_n] = \prod_{i=1}^{n}(1-2p_i).$$

Andererseits kann die Zufallsvariable $y_1 \cdots y_n$ nur die Werte 1 oder -1 annehmen, wir erhalten also:

$$\begin{array}{rcl} \mathbf{E}\,[y_1 \cdot y_2 \cdots y_n] & = & (-1) \cdot \mathbf{Prob}\,(y_1 \cdots y_n = -1) + 1 \cdot (1 - \mathbf{Prob}\,(y_1 \cdots y_n = -1)) \\ & = & 1 - 2 \cdot \mathbf{Prob}\,(y_1 \cdots y_n = -1) \\ \Rightarrow \mathbf{Prob}\,(y_1 \cdots y_n = -1) & = & \dfrac{1 - \mathbf{E}\,[y_1 \cdots y_n]}{2}. \end{array}$$

□

Was hat diese Aussage mit unserer Fragestellung zu tun? Um den Zusammenhang herzustellen, definieren wir $y_i = -1$, falls im i-ten Versuch das Ereignis eintritt, $y_i = +1$ sonst. Da $y_1 \cdots y_n$ nur -1 ist, wenn ungerade viele der y_i den Wert -1 haben, gilt

$$\mathbf{Prob}\,(\text{das Ereignis tritt ungerade oft auf}) = \frac{1 - (1-2p)^n}{2}.$$

1.7 Beispiel. Würfel: $p = \frac{1}{6}, n = 11 \rightarrow \frac{1-(1-\frac{2}{6})^n}{2} \approx 0.49$ ist die Wahrscheinlichkeit, dass bei 11 Würfelexperimenten die Zahl 6 ungerade oft gewürfelt wird.

Fazit: Erwartungswerte können das Leben leichter machen.
Achtung: $\mathbf{E}\,[X_1 \cdot X_2] \neq \mathbf{E}\,[X_1] \cdot \mathbf{E}\,[X_2]$, falls X_1 und X_2 *nicht* unabhängig voneinander sind. Beispiel dazu:

$$\begin{array}{rcl} X_1 & = & \text{Würfelversuch mit 6-seitigem Würfel.} \\ X_2 & = & X_1 \\ \mathbf{E}\,[X_1 \cdot X_2] & = & \frac{1}{6} \cdot (1 \cdot 1) + \frac{1}{6} \cdot (2 \cdot 2) + \cdots + \frac{1}{6} \cdot (6 \cdot 6) \approx 15.17 \end{array}$$

Andererseits ist $\mathbf{E}\,[X_1] \cdot \mathbf{E}\,[X_2] = 3.5 \cdot 3.5 = 12.25$. Übrigens gilt auch für Konstanten c:

- $\mathbf{E}[X + c] = \mathbf{E}[X] + c$
- $\mathbf{E}[X \cdot c] = c \cdot \mathbf{E}[X]$

1.3 Ein einfacher randomisierter Algorithmus für das Problem „Independent Set“

In einem ungerichteten Graphen $G = (V, E)$ mit $|V| = n$ und $|E| = m$ heißt eine Menge $I \subseteq V$ unabhängig (bzw. „Independent Set“), wenn es auf ihr keine Kante aus E gibt.

Es gibt einen (deterministischen) Greedy-Algorithmus, der für ungerichtete Graphen eine unabhängige Menge der Kardinalität mindestens $n/(d_{avg} + 1)$ berechnet, wenn $d_{avg} = \frac{2 \cdot m}{n}$ der Durchschnittsgrad der Knoten im Graphen ist. In diesem Abschnitt wollen wir uns einen randomisierten Algorithmus für Independent Set ansehen, um daran den Umgang mit Erwartungswerten zu üben.

1.8 Behauptung. Sei G ein ungerichteter Graph auf n Knoten mit einem Durchschnittsgrad von $d_{avg} \geq 1$. Es gibt einen randomisierten Algorithmus, der eine unabhängige Menge in G berechnet, die im Erwartungswert mindestens $\frac{n}{2 \cdot d_{avg}}$ viele Knoten enthält.

Beweis: Der Algorithmus arbeitet wie folgt. Er wählt zunächst eine Teilmenge $I \subseteq V$. Dazu startet er mit $I = \emptyset$ und fügt (der Reihe nach) Knoten i mit Wahrscheinlichkeit p_i in die Menge I ein. Nachdem I gewählt wurde, entfernt er für jede Kante, die es auf der Menge I gibt, einen der beiden inzidenten Knoten.
Resultat: $I^* \subseteq V$ ist unabhängige Menge. Wie groß ist $\mathbf{E}[|I^*|]$? Wie viele Knoten haben wir aus I entfernt? Höchstens so viele, wie es Kanten auf I gegeben hat. (Manchmal sogar weniger, wenn z.B. mit einem Knoten mehrere Kanten wegfallen.)

$$\begin{aligned}
\mathbf{E}[|I^*|] &= \mathbf{E}[|I| - \text{Anzahl entfernter Knoten}] \\
&\geq \mathbf{E}[|I| - \text{Anzahl Kanten auf der Menge I}] \\
&= \mathbf{E}[|I|] - \mathbf{E}[\text{Anzahl Kanten auf der Menge I}] \\
&= p_1 + p_2 + \cdots + p_n \; - \sum_{\{i,j\} \in E} p_i \cdot p_j.
\end{aligned}$$

Der letzte Term erklärt sich dabei wie folgt: Eine Kante $\{i, j\}$ „überlebt“, wenn beide Endpunkte in die Menge I aufgenommen worden sind, d.h. mit Wahrscheinlichkeit $p_i \cdot p_j$. Summiert über alle Kanten ergibt sich eine erwartete Kantenanzahl von $\sum_{\{i,j\} \in E} p_i \cdot p_j$.

Die Frage ist nun: Wie wählen wir die p_i? Um den Satz zu beweisen, setzen wir alle p_i auf den gleichen Wert: $\forall\, 1 \leq i \leq n : p_i = p$.
Dann erhalten wir als Ergebnis $\mathbf{E}[|I^*|] \geq p \cdot n - p^2 \cdot m$ und können die rechte Seite maximieren durch die Wahl von $p = \frac{n}{2m} = \frac{1}{d_{avg}}$. Dann ist

$$\mathbf{E}[|I^*|] \geq \frac{n}{d_{avg}} - \frac{n^2}{4m} = \frac{n}{d_{avg}} - \frac{n}{2d_{avg}} = \frac{n}{2d_{avg}}.$$

Als Voraussetzung geht hier ein, dass $d_{avg} \geq 1$ ist, denn nur dann gilt $0 \leq p \leq 1$. □

Andererseits könnte man natürlich die p_i auch anders setzen. Wenn zum Beispiel I_{opt} die größte unabhängige Menge ist, könnte man die p_i als Indikatorvariablen

$$p_i = \begin{cases} 1 & \text{falls } i \in I_{opt} \\ 0 & \text{sonst} \end{cases}$$

wählen. Der Algorithmus wählt dann genau die optimale Menge aus und wir erhalten

$$\mathbf{E}\left[|I^*|\right] = p_1 + \cdots + p_n \; - \sum_{\{i,j\}\in E} p_i \cdot p_j = |I_{opt}|,$$

da I_{opt} eine unabhängige Knotenmenge ist und somit $p_i \cdot p_j = 0$ für alle $\{i,j\} \in E$ gilt.
Die Schwierigkeit bei diesem Ansatz ist, dass die optimale Wahl der p_i genau so schwer wie Independent Set selbst und damit NP-hart ist. Nächster Ansatz: Wähle die Wahrscheinlichkeit abhängig vom Grad

$$p_i = \frac{1}{d_i + 1}.$$

Knoten mit niedrigem Grad werden so wahrscheinlicher gewählt als Knoten mit hohem Grad. Übungsaufgabe: Welche Aussage bekommt man mit dieser Wahl?

1.4 Eine Analyse von randomisiertem Quicksort

Da wir alle randomisiertes Quicksort kennen und uns in diesem Abschnitt nur die Analyse interessiert, gehen wir davon aus, dass eine Menge S von Zahlen zu sortieren ist, die alle voneinander verschieden sind. (Die Menge S ist dabei als Array implementiert, das alle Elemente der Menge enthält.)

Algorithmus `RandQSort`(Menge S)

1. Falls $|S| = 1$, dann gib S zurück.
2. Wähle das Splitelement $s \in S$ zufällig gemäß Gleichverteilung.
3. Berechne $S_<$ und $S_>$, die Mengen der kleineren und größeren Elemente, also

   $$S_< := \{x \in S \mid x < s\} \text{ und } S_> := \{x \in S \mid x > s\}.$$

   Hierbei wird s also mit allen anderen Elementen aus S verglichen.
4. Die sortierte Reihenfolge ist (`RandQSort`$(S_<)$, s, `RandQSort`$(S_>)$).

1.9 Satz. `RandQSort` *macht im Erwartungswert* $2(n+1)H_n - 4n = O(n \log n)$ *viele Vergleiche. Dabei ist* H_n *die* n*-te Harmonische Zahl.*

Beweis: O.B.d.A. nehmen wir an, dass S (bzw. die Arraydarstellung von S) eine zufällige Permutation der Menge $\{1, \ldots, n\}$ ist. Wir definieren Indikatorvariablen

$$X_{i,j} = \begin{cases} 1, & \text{falls } i \text{ und } j \text{ im Algorithmus miteinander verglichen werden} \\ 0 & \text{sonst} \end{cases}.$$

Die Anzahl der Vergleiche insgesamt beträgt dann $\sum_{i<j} X_{i,j}$. Dies gilt, weil kein Vergleich zweimal vorgenommen wird, wie man sich leicht überlegt. Wir berechnen

$$\mathbf{E}\left[\sum_{i<j} X_{i,j}\right] = \sum_{i<j} \mathbf{E}\left[X_{i,j}\right] = \sum_{i<j} p(i,j),$$

wobei $p(i,j)$ die Wahrscheinlichkeit ist, dass i und j verglichen werden. Die letzte Gleichung gilt, da der Wert von $\mathbf{E}\left[X_{i,j}\right]$ der Wahrscheinlichkeit entspricht, dass $X_{i,j} = 1$ ist.

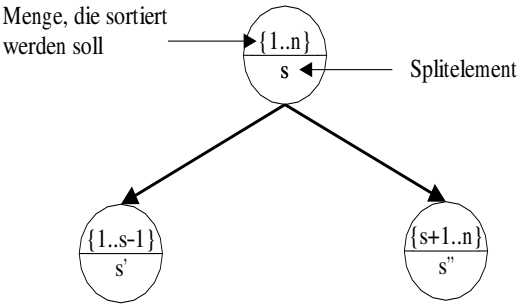


Abbildung 1.1: Ausschnitt des Rekursionsbaumes von `RandQSort`.

Man kann den Verlauf des Algorithmus als Baum darstellen. In jedem Knoten steht die Grundmenge und das Splitelement. So wird die Menge $\{1,\ldots,n\}$ durch das Splitelement s in $\{1,\ldots,s-1\}$ und $\{s+1,\ldots,n\}$ unterteilt.
Wir betrachten einen Knoten, der die Zahlen i und j und damit auch alle Zahlen aus dem Intervall $[i,j]$ enthält, und unterscheiden drei Fälle:

1. **Fall:** Das Splitelement s des Knotens liegt nicht im Intervall $[i,j]$. Dann ist das Intervall $[i,j]$ vollständig in $S_>$ oder vollständig in $S_<$ enthalten. Daher wird in diesem Knoten noch nicht entschieden, ob i und j miteinander verglichen werden.

2. **Fall:** Es ist $s=i$ oder $s=j$. Wenn eine der beiden Grenzen Splitelement ist, wird der Vergleich zwischen i und j auf jeden Fall durchgeführt.

3. **Fall:** Es gilt $i<s<j$. Liegt das Splitelement innerhalb des Intervalls, wird das Intervall an dieser Stelle aufgespalten. Daher landen i und j in disjunkten Teilmengen (i in $S_<$ und j in $S_>$) und werden nie verglichen.

Die Entscheidung, ob i und j miteinander verglichen werden, fällt also in dem Knoten, in dem das Splitelement s eine Zahl aus dem Intervall $[i,j]$ ist. Nur dann, wenn $s=i$ oder $s=j$ ist, werden i und j miteinander verglichen und sonst nicht. Also ist

$$p(i,j)=\frac{2}{j-i+1} \Rightarrow \mathbf{E}\,[\text{Anzahl Vergleiche}]=\sum_{i<j}\frac{2}{j-i+1}.$$

Es gibt ein Paar (i,j) mit $j-i+1=n$, nämlich $i=1, j=n$. Es gibt zwei Paare (i,j) mit $j-i+1=n-1$, nämlich $i=1, j=n-1$ und $i=2, j=n$. Entsprechend gibt es $n-1$ Paare (i,j) mit $j-i+1=2$. Wir erhalten:

$$\begin{aligned}\sum_{i<j}\frac{2}{j-i+1} &= \frac{2}{2}\cdot(n-1)+\frac{2}{3}\cdot(n-2)+\cdots+\frac{2}{n}\cdot 1\\ &= \sum_{i=2}^{n}\frac{2}{i}(n+1-i)=2(n+1)\sum_{i=2}^{n}\frac{1}{i}-\sum_{i=2}^{n}2\\ &= 2(n+1)(H_n-1)-2(n-1)=2(n+1)H_n-4n.\end{aligned}$$

□

1.5 Wichtige Ungleichungen: Markov, Tschebyscheff, Jensen

1.10 Satz (Markov)**.** *Sei X eine Zufallsvariable, die nur nichtnegative Werte annimmt, und sei $a > 0$. Dann gilt:*

$$\mathbf{Prob}\,(X \geq a) \leq \frac{\mathbf{E}\,[X]}{a}.$$

Beweis:

$$\begin{aligned}
\mathbf{E}\,[X] &= \sum_{x \in M} \mathbf{Prob}\,(X = x) \cdot x \\
&\geq \sum_{\substack{x \in M \\ x \geq a}} \mathbf{Prob}\,(X = x) \cdot x \\
&\geq \sum_{\substack{x \in M \\ x \geq a}} \mathbf{Prob}\,(X = x) \cdot a \\
&= a \cdot \sum_{\substack{x \in M \\ x \geq a}} \mathbf{Prob}\,(X = x) \\
&= a \cdot \mathbf{Prob}\,(X \geq a)
\end{aligned}$$

Also gilt $\mathbf{Prob}\,(x \geq a) \leq \frac{\mathbf{E}[X]}{a}$. Die Nichtnegativität der Werte geht hier wesentlich in den Beweis ein. □

Beachte: Für $a \leq \mathbf{E}\,[X]$ ist die Aussage des Satzes trivial.

Anwendung: Man kann mit der Markovschen Ungleichung die Wahrscheinlichkeit eingrenzen, mit der Zufallsvariablen um ein Vielfaches größer als ihr Erwartungswert sind. Zum Beispiel folgt $\mathbf{Prob}\,(X \geq c \cdot \mathbf{E}\,[X]) \leq 1/c$.

1.11 Korollar. Sei $g : \mathbf{R}_0^+ \to \mathbf{R}^+$ eine streng monoton wachsende Funktion und X eine Zufallsvariable, die nur nichtnegative Werte annimmt. Dann gilt für alle $a > 0$:

$$\mathbf{Prob}\,(X \geq a) \leq \frac{\mathbf{E}\,[g(X)]}{g(a)}.$$

Beweis: $g(X)$ ist eine Zufallsvariable, die nur nichtnegative Werte annimmt und es gilt: $g(a) > 0$ aufgrund des Wertebereichs. Aus der strengen Monotonie folgt $X \geq a \Leftrightarrow g(X) \geq g(a)$ und mit Anwendung der Markovschen Ungleichung folgt:

$$\mathbf{Prob}\,(X \geq a) = \mathbf{Prob}\,(g(X) \geq g(a)) \leq \frac{\mathbf{E}\,[g(X)]}{g(a)}.$$

□

Achtung! Typischerweise ist $\mathbf{E}\,[g(X)] \neq g(\mathbf{E}\,[X])$. (Beispiel: $g(X) = X \cdot X$.)

1.12 Satz (Tschebyscheff)**.** *Sei X eine Zufallsvariable und $\mathbf{V}\,[X] = \mathbf{E}\left[(X - \mathbf{E}\,[X])^2\right]$ die Varianz von X. Wenn $\mathbf{V}\,[X] > 0$ ist, dann gilt für alle $t > 0$:*

$$\mathbf{Prob}\left(|X - \mathbf{E}\,[X]\,| \geq t \cdot \sqrt{\mathbf{V}\,[X]}\right) \leq \frac{1}{t^2}.$$

Beweis:

$$\begin{aligned}
\mathbf{Prob}\left(|X - \mathbf{E}\,[X]\,| \geq t \cdot \sqrt{\mathbf{V}\,[X]}\right) &= \mathbf{Prob}\left((X - \mathbf{E}\,[X])^2 \geq t^2 \cdot \mathbf{V}\,[X]\right) \\
&\overset{Markov}{\leq} \frac{\mathbf{E}\left[(X - \mathbf{E}\,[X])^2\right]}{t^2 \cdot \mathbf{V}\,[X]} = \frac{1}{t^2}.
\end{aligned}$$

□

1.13 Definition. Eine reellwertige Funktion f heißt *konkav* auf einem Intervall I, falls für alle $0 \leq \alpha \leq 1$ und alle $x \neq y \in I$ gilt:

$$f(\alpha \cdot x + (1-\alpha) \cdot y) \geq \alpha \cdot f(x) + (1-\alpha) \cdot f(y).$$

f heißt *streng* konkav auf einem Intervall I, falls obige Ungleichung mit „$>$“ statt „$\geq$“ gilt.

1.14 Satz (Jensen)**.** *Sei f eine stetige streng konkave Funktion auf dem Intervall I und*

$$\sum_{i=1}^{n} a_i = 1,$$

mit $a_i > 0$ für $1 \leq i \leq n$. Dann gilt für alle $x_i \in I$

$$\sum_{i=1}^{n} a_i f(x_i) \leq f\left(\sum_{i=1}^{n} a_i x_i\right).$$

Die beiden Summen sind genau dann gleich, wenn $x_1 = \cdots = x_n$ gilt.

1.15 Korollar. Sei X eine Zufallsvariable mit endlicher Trägermenge und f eine stetige streng konkave Funktion. Dann ist

$$\mathbf{E}\left[f(X)\right] \leq f(\mathbf{E}\left[X\right]).$$

1.6 Las-Vegas- und Monte-Carlo-Algorithmen

Die Verwendung von Zufallszahlen kann zwei mögliche Auswirkungen haben:

- die Laufzeit ist zufällig
- das Ergebnis ist zufällig

Man unterteilt die Algorithmen daher in zwei Klassen:

Las Vegas: Die Laufzeit darf zufällig sein, das Ergebnis ist aber immer richtig.

Monte Carlo: Die Laufzeit darf zufällig sein, das Ergebnis ebenso. Ein Monte-Carlo-Algorithmus kann falsche oder nicht optimale Ergebnisse liefern; je nach Anwendungsgebiet ist daher evtl. noch eine Prüfung auf Korrektheit oder ein weiterer Durchlauf nötig.

Es gilt die Hofmeistersche Merkregel:

> Monte Carlo → MC → „mostly correct“, also nicht immer korrekt, also fehlerbehaftet.
> Las Vegas → LV → „Laufzeit variabel“.

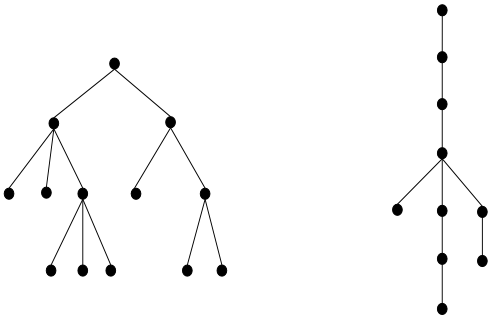


Abbildung 1.2: Zwei Darstellungsarten für den Ablauf randomisierter Algorithmen. Links Knoten mit variabler, rechts mit konstanter Laufzeit.

Effizienz und erwartete Laufzeit

Da Monte-Carlo-Algorithmen falsche Ergebnisse liefern können, stellt man bei ihnen schärfere Forderungen an die Laufzeit:

Monte-Carlo-Algorithmen heißen effizient, wenn ihre worst-case-Laufzeit polynomiell ist.

Las-Vegas-Algorithmen heißen effizient, wenn ihre erwartete Laufzeit polynomiell ist.

Beispiel: `RandQSort` ist ein effizienter Las-Vegas-Algorithmus.

Den Ablauf eines randomisierten Algorithmus kann man sich durch einen Baum veranschaulichen. In den Knoten geht der Algorithmus deterministisch vor und immer dann, wenn der Ausgang eines Zufallsexperimentes den Verlauf des Algorithmus bestimmt, verzweigt sich der Knoten je nach Ausgang des Experimentes.
Die Laufzeit kann in jedem Knoten unterschiedlich sein; alternativ kann man auch Ketten, d.h. Knoten vom Grad 1 zulassen. In diesem Fall hat jeder Knoten konstante Laufzeit.

Eine Definition der erwarteten Laufzeit über Blätter dieses Baumes ist „gefährlich“, da es unendliche Pfade geben kann. Man definiert daher die erwartete Laufzeit durch

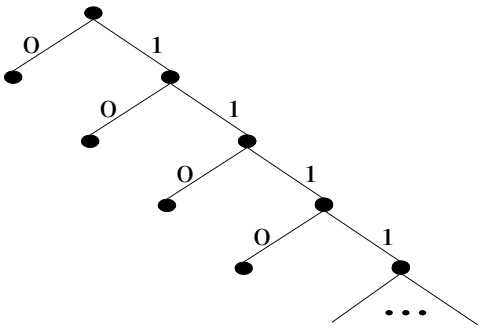
$$\sum_{i \text{ Knoten des Baumes}} \textbf{Prob}\,(i \text{ wird erreicht}) \cdot \text{Laufzeit im Knoten } i.$$

Bei Algorithmen, die mit positiver Wahrscheinlichkeit in eine Endlosschleife geraten, ergibt sich nach dieser Definition eine unendliche erwartete Laufzeit. (So, wie es sein sollte.)

Algorithmus A:

repeat
    wirf Münze $\in \{0, 1\}$, je mit Wahrscheinlichkeit $\frac{1}{2}$
until Münze $= 0$

Der zugehörige Baum sieht folgendermaßen aus:



Angenommen, jeder Knoten hat Kosten 1. Wir benötigen Kosten 1, um die Wurzel zu bearbeiten. Auf jeder folgenden Ebene i befinden sich 2 Knoten, die jeweils mit Wahrscheinlichkeit $\left(\frac{1}{2}\right)^i$ erreicht werden. Damit beträgt die erwartete Laufzeit

$$1 + \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i \cdot 2 \cdot 1 = 1 + 2\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i = 1 + 2 = 3.$$

Betrachten wir nun den Fall einer „schiefen“ Münze:

Algorithmus B:

repeat
    wirf Münze $\in \{0, 1\}$, 0 mit W'keit p, 1 mit W'keit $1 - p$
until Münze $= 0$

In diesem allgemeineren Fall werden die 0-Knoten und die 1-Knoten mit unterschiedlicher Wahrscheinlichkeit erreicht: auf Ebene i wird der 0-Knoten mit Wahrscheinlichkeit $(1-p)^{i-1} \cdot p$ erreicht, der 1-Knoten mit Wahrscheinlichkeit $(1-p)^i$. Die erwartete Laufzeit beträgt

$$1 + \sum_{i=1}^{\infty} \left((1-p)^{i-1} \cdot p + (1-p)^i\right).$$

Mit der bekannten Summenformel $\sum_{i=0}^{\infty} q^i = 1/(1-q)$ für $0 < q < 1$ erhalten wir als Laufzeit:

$$1 + (1/p) \cdot p + (1/p - 1) = (1/p) + 1.$$

Da die Rechenzeit bei unserem Algorithmus um 1 größer ist als die Anzahl der gemachten Versuche, haben wir außerdem erhalten, dass bei Wahrscheinlichkeit p für das Ereignis die durchschnittliche Anzahl an Versuchen, bis dieses Ereignis eintritt, $1/p$ ist. Exemplarisch wollen wir dies bei der Analyse des folgenden einfachen Algorithmus zum zufälligen Auswürfeln von Lottoreihen benutzen:

```
for i=1 to 49 do a[i]=0;
for i=1 to 6 do
    begin
      repeat würfle zahl aus {1,...,49}
      until a[zahl]=0;
      a[zahl]=1;
    end:
for i = 1 to 49 do if a[i]=1 then output[i];
```

Wieviele Würfelversuche finden im Erwartungswert statt? Bei $i = 1$ ist die Wahrscheinlichkeit, direkt $a[zahl] = 0$ zu haben, gleich 1. Die erwartete Anzahl an Versuchen ist also auch 1. Für $i = 2$ ist diese Wahrscheinlichkeit nur noch $\frac{48}{49}$, also haben wir hier eine erwartete Anzahl von $\frac{49}{48}$ Versuchen. Allgemein ist für i diese Wahrscheinlichkeit $\frac{50-i}{49}$ und die erwartete Anzahl Versuche $\frac{49}{50-i}$. Summiert man über alle i, so erhält man insgesamt eine erwartete Anzahl von $\frac{49}{49} + \frac{49}{48} + \cdots + \frac{49}{44} \approx 6.33$ Versuchen.

1.7 Komplexitätsklassen

Wir wollen in diesem Abschnitt Komplexitätsklassen zu randomisierten Algorithmen vorstellen. Um die Klassen übersichtlicher beschreiben zu können, verwenden wir folgende Notation:
$M(x) = 1$ soll heißen, dass der Algorithmus M das Wort x akzeptiert, $L(x) = 1$ soll heißen, dass das Wort x in der Sprache L enthalten ist. Entsprechend heißt $M(x) = 0$, dass das Wort nicht akzeptiert wird bzw. $L(x) = 0$, dass das Wort nicht in der Sprache enthalten ist. Wir beginnen mit einer prominenten deterministischen Klasse:

1.16 Definition (P — polynomial time)**.** Die Klasse P enthält alle Sprachen L, für die ein deterministischer Algorithmus M existiert, der die folgenden Eigenschaften hat:

a) Auf jeder Eingabe x ist die Laufzeit polynomiell.

b) Für alle Eingaben x ist $M(x) = L(x)$.

Da P nur deterministische Algorithmen erlaubt und außerdem eine altbekannte Klasse ist, gehen wir hier nicht näher auf sie ein.

1.17 Definition (ZPP — zero error probabilistic polynomial time)**.** Die Klasse ZPP enthält alle Sprachen L, für die ein randomisierter Algorithmus M existiert, der die folgenden Eigenschaften hat:

a) Auf jeder Eingabe x ist die *erwartete* Laufzeit polynomiell.

b) Für alle Eingaben x ist $\mathbf{Prob}\,(M(x) = L(x)) = 1$.

Ein Algorithmus aus ZPP heißt auch (effizienter) *Las-Vegas-Algorithmus*.

1.18 Definition (RP — randomized polynomial time)**.** Die Klasse RP enthält alle Sprachen L, für die ein randomisierter Algorithmus M existiert, der die folgenden Eigenschaften hat:

a) Auf jeder Eingabe x ist die worst-case-Laufzeit polynomiell.

b) $x \in L \Rightarrow \mathbf{Prob}\,(M(x) = L(x)) \geq \frac{1}{2}$.

$x \notin L \Rightarrow \mathbf{Prob}\,(M(x) = L(x)) = 1$.

Solch ein Algorithmus heißt *one-sided error Monte-Carlo-Algorithmus*.

Die Erfolgswahrscheinlichkeit kann durch wiederholtes Anwenden *amplifiziert* werden:

Algorithmus $B(x)$
for $i := 1$ **to** k **do if** $M(x) = 1$ **then** akzeptiere x und STOPP.
Verwirf x.

Für $x \notin L$ ist stets $M(x) = 0$ und somit gibt Algorithmus B auf x eine 0 aus.
Für $x \in L$ sei $p_x := \mathbf{Prob}\,(M(x) = 1)$. Dann folgt: $\mathbf{Prob}\,(B(x) = 1) = 1 - (1 - p_x)^k$. Man sagt, dass p_x amplifiziert wird zu $1 - (1 - p_x)^k$. Als Beispiel (bei $k = 2$): $p_x = \frac{1}{2}$ wird amplifiziert zu $\frac{3}{4}$ und $p_x = 0$ wird amplifiziert zu 0.

1.19 Definition (BPP – bounded error probabilistic polynomial time)**.** Die Klasse BPP enthält alle Sprachen L, für die ein randomisierter Algorithmus M existiert, der die folgenden Eigenschaften hat:

a) Auf jeder Eingabe x ist die worst-case-Laufzeit polynomiell.

b) Für jede Eingabe x gilt: $\mathbf{Prob}\,(M(x) = L(x)) \geq \frac{3}{4}$.

1.20 Definition (Die Klasse PP — probabilistic polynomial time)**.** Die Klasse PP enthält alle Sprachen L, für die ein randomisierter Algorithmus M existiert, der die folgenden Eigenschaften hat:

a) Für jede Eingabe x ist die worst-case-Laufzeit polynomiell.

b) Für jede Eingabe x gilt: $\mathbf{Prob}\,(M(x) = L(x)) > \frac{1}{2}$.

Aus der Definition folgt übrigens direkt, dass BPP $\subseteq$ PP ist.
Wir wollen nun untersuchen, was passiert, wenn wir einen PP-Algorithmus M iterieren. Zunächst ein einfacher konkreter Fall, dann der allgemeine Fall.

Algorithmus B Wende Algorithmus M dreimal an und gib die häufigere Antwort als Antwort aus. Es folgt: $\mathbf{Prob}\,(B(x) = 1) = \binom{3}{2} \cdot p_x^2 \cdot (1 - p_x) + p_x^3 = p_x^2 \cdot (3 - 2p_x)$. Ein Beispiel: $p_x = 0.51$ wird amplifiziert zu ≈ 0.515 und $p_x = 0.49$ wird amplifiziert zu ≈ 0.485. Der Wert $p_x = \frac{1}{2}$ wird amplifiziert zu $\frac{1}{2}$. Allgemein erhalten wir:

1.7.1 Amplifikation bei PP-Algorithmen

Gegeben sei ein PP-Algorithmus M mit

$$\mathbf{Prob}\,(M(x) = L(x)) \geq p$$

für alle Eingaben x.
Wir iterieren den Algorithmus M genau T-mal und treffen eine Mehrheitsentscheidung (T ungerade). Wir arbeiten bei der Analyse des „neuen“ Algorithmus mit Indikatorvariablen X_i, $i = 1, \ldots, T$. Es sei $X_i = 1$, falls der Algorithmus im i-ten Durchlauf einen Fehler macht und $X_i = -1$, falls er richtig rechnet. Es gilt nun:

$$\mathbf{Prob}\,(\text{neuer Algorithmus macht Fehler}) = \mathbf{Prob}\left(\sum_{i=1}^{T} X_i \geq 0\right).$$

(Man beachte dabei, dass wegen der Wahl von T als ungerade Zahl der Fall $\sum_{i=1}^{T} X_i = 0$ gar nicht eintreten kann.) Definiere nun eine neue ZV $Y := c^{X_1 + \cdots + X_T}$ $(c > 1)$. Es folgt:

$$\begin{aligned}
\mathbf{Prob}\left(\sum_{i=1}^{T} X_i \geq 0\right) &= \mathbf{Prob}\,(Y \geq 1) \quad \text{(strenge Monotonie)} \\
&\overset{\text{Markov}}{\leq} \mathbf{E}\,[Y] = \mathbf{E}\left[c^{X_1 + \cdots + X_T}\right] = \mathbf{E}\left[c^{X_1} \cdots c^{X_T}\right] \\
&= \mathbf{E}\left[c^{X_1}\right] \cdots \mathbf{E}\left[c^{X_T}\right] \quad \text{(wegen Unabhängigkeit der } X_i\text{)} \\
&= \mathbf{E}\left[c^{X_1}\right]^T
\end{aligned}$$

Mit der Abkürzung $p_1 = \mathbf{Prob}\,(X_1 = -1)$ ist $\mathbf{E}\left[c^{X_1}\right] = c \cdot (1 - p_1) + c^{-1} \cdot p_1$. Dies ist monoton fallend in p_1, wir erhalten also wegen $p_1 \geq p$:

$$\mathbf{E}\left[c^{X_1}\right] \leq c \cdot (1 - p) + c^{-1} \cdot p.$$

Insgesamt macht der Algorithmus also einen Fehler mit Wahrscheinlichkeit höchstens $\left[c \cdot (1 - p) + \frac{1}{c} \cdot p\right]^T$. Wenn wir nun $c = \frac{1}{2(1-p)} > 1$ wählen, dann ist

$$\mathbf{Prob}\,(\text{neuer Algorithmus macht Fehler}) \leq \left(\frac{1}{2} + 2 \cdot p \cdot (1 - p)\right)^T.$$

(Anmerkung zur Wahl von c: Eigentlich würde man das c so wählen, dass $c \cdot (1-p) + (1/c) \cdot p$ bei vorgegebenem p möglichst klein wird. Dazu müsste man, wie aus der Analysis bekannt ist, die Ableitung berechnen etc., und man käme zu einer anderen Wahl von c. Der Grund dafür, dass wir nicht das optimale c wählen, ist, dass wir die Rechnung auch erträglich kurz halten wollen.)

Für $p = \frac{1}{2} + \varepsilon$ ergibt sich

$$\begin{aligned} \textbf{Prob}\,(\text{neuer Algorithmus macht Fehler}) &\leq \left(\frac{1}{2} + 2\left(\frac{1}{4} - \varepsilon^2\right)\right)^T \\ &= \left(1 - 2\varepsilon^2\right)^T. \end{aligned}$$

Wir wollen erreichen, dass gilt: $\left(1-2\varepsilon^2\right)^T \leq 2^{-k}$. Wähle dazu $T \geq \frac{k \ln 2}{2\varepsilon^2}$. Es folgt:

$$\left(1-2\varepsilon^2\right)^T \leq \left(1-2\varepsilon^2\right)^{\frac{k\ln 2}{2\varepsilon^2}} \leq \left(e^{-2\varepsilon^2}\right)^{\frac{k\ln 2}{2\varepsilon^2}} = e^{-k\ln 2} = 2^{-k}.$$

Wir sehen: Wenn $1/\varepsilon$ durch ein Polynom in n beschränkt ist, (also ε nicht zu klein ist), dann kann man aus dem PP-Algorithmus einen BPP-Algorithmus machen.

Offene Probleme:

- RP = co-RP ?
- RP $\subseteq$ NP $\cap$ co-NP ?
- BPP $\subseteq$ NP ?

1.7.2 Komplexitätsklassen und Algorithmen

Intuitiv werden die Bezeichnungen für Komplexitätsklassen auch auf Algorithmen ausgedehnt, beispielsweise bezeichnet man den randomisierten Quicksortalgorithmus auch als ZPP-Algorithmus. Bei booleschen Funktionen $f : \{0,1\}^* \to \{0,1\}$ ist eine Übertragung wie folgt möglich:

$$f \in \text{ZPP} :\Leftrightarrow f^{-1}(1) \in \text{ZPP},$$

analog natürlich auch für die anderen Komplexitätsklassen. Das folgende Beispiel soll aber davor warnen, allzu leichtfertig mit solchen Sprechweisen umzugehen. Natürlich könnte man sich auch bei Optimierungsproblemen die Frage stellen, ob es ZPP-Algorithmen oder RP-Algorithmen für diese gibt oder nicht. Wie könnte man aber definieren, wann ein Optimierungsproblem $OPTP$ in ZPP ist? Es gibt zwei unterschiedliche nahe liegende Ansätze dafür.

Beispiel: Wir betrachten ein Maximierungsproblem $OPTP$ und definieren wie folgt zwei boolesche Funktionen:
$f_k(x) = 1 \quad :\Leftrightarrow \quad OPTP(x) = k$
$f_k^*(x) = 1 \quad :\Leftrightarrow \quad OPTP(x) \geq k$

1.21 Behauptung. Wenn wir einen randomisierten Polynomialzeitalgorithmus A für $OPTP$ haben, der auf jeder Eingabe x mit Wahrscheinlichkeit mindestens ε die korrekte Ausgabe $OPTP(x)$ liefert und sonst nur Werte kleiner als $OPTP(x)$, dann ist $f_k \in BPP$ und $f_k^* \in RP$.

Beweis: Wir modifizieren Algorithmus A (durch hinreichend häufige Iteration), so dass $\varepsilon \geq 3/4$ gilt. Weiter definieren wir zwei weitere Algorithmen:

Algorithmus B	Falls $A(x) = k$, gib 1 aus, sonst 0.
Algorithmus B*	Falls $A(x) \geq k$, gib 1 aus, sonst 0.

Analyse zu Algorithmus B:
Wenn $f_k(x) = 1$ ist, dann ist $OPTP(x) = k$, also liefert A mit Wahrscheinlichkeit mindestens $3/4$ die Ausgabe k, also ist $\mathbf{Prob}\,(B(x) = 1) \geq 3/4$.

Wenn $f_k(x) = 0$ ist, dann wissen wir nur, dass $OPTP(x) \neq k$ ist. Zwangsläufig ergeben sich zwei Fälle:
Fall I) $OPTP(x) < k$. Dann liefert A nur Werte, die kleiner als k sind, also ist $B(x) = 0$.
Fall II) $OPTP(x) > k$. Dann liefert A mit Wahrscheinlichkeit mindestens $3/4$ den Wert $OPTP(x) > k$, also ist $\mathbf{Prob}\,(B(x) = 0) \geq 3/4$.
Insgesamt ist also $\mathbf{Prob}\,(B(x) = f_k(x)) \geq 3/4$, B ist also ein BPP-Algorithmus, und wir haben einen zweiseitigen Fehler.

Analyse zu Algorithmus B^*:
Falls $f_k^*(x) = 1$, folgt $OPTP(x) \geq k$, also ist $\mathbf{Prob}\,(A(x) \geq k) \geq 3/4$, also ist $\mathbf{Prob}\,(B^*(x) = 1) \geq 3/4$.
Falls $f_k^*(x) = 0$, ist $OPTP(x) < k$, also liefert A nur Werte kleiner als k, also ist $\mathbf{Prob}\,(B^*(x) = 1) = 0$.
Demnach ist B^* ein RP-Algorithmus, er hat also nur einseitigen Fehler. □

Sollte man nun sagen, dass das gegebene Optimierungsproblem in RP ist, oder sollte man sagen, dass es in BPP liegt?

1.8 Randomisierte Schaltkreise

Das einfachste Modell für parallele Algorithmen ohne Berücksichtigung der Interprozesskommunikation ist der **Schaltkreis**, also ein azyklischer gerichteter Graph, dessen Knoten Gatter, Variablen, negierte Variablen oder Konstanten repräsentieren. Die Gatter berechnen boolesche Funktionen, der Schaltkreis insgesamt berechnet eine boolesche Funktion von $\{0,1\}^n$ nach $\{0,1\}$. Die **Größe** eines Schaltkreises ist definiert als die Anzahl seiner Gatter. Für probabilistische Algorithmen wird dieses Modell zum **probabilistischen Schaltkreis** erweitert, indem zusätzlich zu den n Inputvariablen $x = (x_1, \ldots, x_n)$ noch N Zufallsvariablen $r = (r_1, \ldots, r_N)$ in die Berechnung eingehen, die jeweils mit Wahrscheinlichkeit $1/2$ den Wert 0 oder 1 annehmen können. Das Beispiel aus Abbildung 1.3 soll diese Definition verdeutlichen. Wenn $x_1 = 1$ ist, dann lautet das Ergebnis 1. Für $x_1 = 0$ wird jeweils mit Wahrscheinlichkeit $1/2$ der Wert 0 oder 1 ausgegeben.

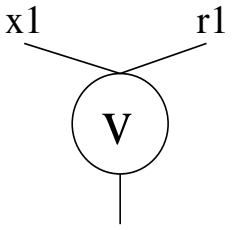


Abbildung 1.3: Probabilistisches ODER-Gatter.

Die durch einen probabilistischen Schaltkreis berechnete Funktion wird durch folgende Definition erfasst:

1.22 Definition. Ein probabilistischer Schaltkreis S mit den Zufallseingaben $r = (r_1, \ldots, r_N)$ berechnet eine boolesche Funktion f, wenn für alle Eingaben x gilt:

$$\begin{aligned} f(x) = 0 \quad &\Rightarrow \quad \mathbf{Prob}\,(S(x,r) = 0) = 1 \\ f(x) = 1 \quad &\Rightarrow \quad \mathbf{Prob}\,(S(x,r) = 1) \geq 1/2. \end{aligned}$$

(**Randbemerkung:** Wenn man mit Hilfe zweier UND-Gatter die Funktion $x_1 \wedge r_1 \wedge r_2$ baut, dann berechnet dieser probabilistische Schaltkreis gemäß Definition keine Funktion, da auf dem Input $x_1 = 1$ mit Wahrscheinlichkeit genau $1/4$ eine 1 ausgegeben wird.)

Damit gilt: Wenn ein Schaltkreis S die boolesche Funktion f berechnet, dann folgt aus $S(x, r) = 1$, dass $f(x) = 1$ ist. Man nennt r dann auch einen *Zeugen* für $f(x) = 1$.

Dass mit probabilistischen Schaltkreisen polynomieller Größe kein Gewinn bei der Berechnungskraft gegenüber deterministischen Schaltkreisen erzielt werden kann, zeigt der folgende Satz.

1.23 Satz. *Sei S ein probabilistischer Schaltkreis der Größe T, der die Funktion $f(x_1, \ldots, x_n)$ berechnet. Dann gibt es einen deterministischen Schaltkreis für f, der höchstens die Größe $T \cdot (n + 1) + 1$ hat.*

Beweis: Wir konstruieren eine Matrix, deren m Zeilen den Inputs entsprechen, auf denen f die Ausgabe 1 berechnet, und deren Spalten allen Belegungen des Vektors r der Zufallsvariablen entsprechen. Zur Notation: Die Indizierung $x^{(i)}$ bezeichne die i-te Belegung des Vektors x mit $f(x) = 1$, analog bezeichne $r^{(j)}$ die j-te Belegung des Vektors r in der kanonischen Reihenfolge. Demnach hat die Matrix maximal 2^n Zeilen und genau 2^N Spalten. Die Matrixeinträge korrespondieren zu den Berechnungsergebnissen des Schaltkreises: An der Position (i, j) steht der Eintrag $S(x^{(i)}, r^{(j)})$.
Wesentlich ist nun die Beobachtung:

> In jeder Zeile ist die Anzahl der Einsen immer mindestens so groß wie die Anzahl der Nullen. $(*)$

Dies ergibt sich aus der Tatsache, dass S die Funktion f berechnet. Nach dem Schubfachprinzip gibt es eine Spalte, die mindestens so viele Einsen wie Nullen enthält: Da es $m2^N$ Einträge gibt, gibt es mindestens $\frac{m2^N}{2}$ Einsen, und daher eine Spalte mit mindestens $m/2$ Einsen. Der zugehörige r-Vektor dieser Spalte ist demnach Zeuge für mindestens die Hälfte der Inputs mit $f(x) = 1$.
Also reicht es aus, sich den Zeugen zu merken und danach die entsprechende Spalte und alle Zeilen, in denen die Matrix in der entsprechenden Spalte eine Eins hat, zu streichen. Die Restmatrix genügt immer noch der Invariante $(*)$, weil anschaulich gesprochen in den relevanten Zeilen immer nur die Spalten mit Nullen gestrichen wurden.
Damit kann das Vorgehen rekursiv fortgesetzt werden. Zu Beginn gibt es $a \leq 2^n$ Zeilen, nach einer Iteration des Algorithmus höchstens $a/2$ Zeilen, und so weiter. Also gibt es nach (spätestens) $n+1$ Iterationen höchstens $a/2^{n+1} < 1$ Zeilen, und es ist keine Zeile mehr vorhanden.
Wir haben nun eine Zeugenmenge R erhalten, und für diese gilt, dass jeder Input x mit $f(x) = 1$ einen Zeugen in R hat. Wir wählen als deterministischen Schaltkreis den Schaltkreis

$$S^*(x) := \bigvee_{r \in R} S(x, r).$$

Dieser berechnet die Funktion f, weil es für Eingaben, die auf 0 abgebildet werden, keinen Zeugen gibt, und weil es für alle Eingaben, die auf 1 abgebildet werden, nach Konstruktion einen Zeugen in der Menge R gibt.
Eine obere Schranke für die Größe des resultierenden Schaltkreises erhalten wir einfach über die Summe der Größen der benutzten Schaltkreise und deren „Veroderung“, $|R| \cdot T + 1 \leq (n+1) \cdot T + 1$. □

Dieses Vorgehen zeigt, dass Randomisierung prinzipiell beseitigt werden kann. Im obigen Beispiel bräuchte man dafür allerdings exponentielle Rechenzeit, denn wie will man sonst die geeigneten Zeugen finden?

Kapitel 2

Grundlegende Beispiele für randomisierte Algorithmen

2.1 Hidden-Line-Elimination, binäre planare Partitionen

Hierbei handelt es sich um ein Beispiel aus der Computergrafik. Der Originalartikel „Paterson/Yao (1990): *Efficient binary space partitions for hidden-surface removal and solid modeling,* Discrete & Computational Geometry 5:485-503“ enthält insbesondere den für die Praxis relevanteren 3D-Fall; hier betrachten wir nur den 2D-Fall.

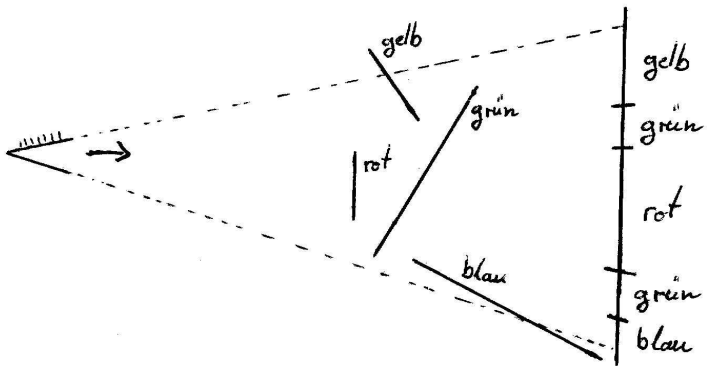


Abbildung 2.1: Beobachter und Objekte im 2-dimensionalen Raum.

Der Beobachter wandert, die Objekte bleiben an gleicher Stelle (siehe Abbildung 2.1). Es ist also ein langsames Preprocessing erlaubt und ein schneller neuer Aufbau gewünscht. Bei dem folgenden Ansatz werden so genannte „binäre planare Partitionen“ benutzt.

2.1 Definition. Eine *binäre planare Partition* ist gegeben durch einen Baum, in dem alle Knoten genau zwei Kinder oder kein Kind haben, und in dem folgendes gilt:

- Jeder Knoten des Baums beschreibt eine konvexe Teilmenge des $\mathbf{R}^2$.
- Die Wurzel beschreibt $\mathbf{R}^2$.
- Wenn ein Knoten zwei Kinder hat und die Teilmenge M beschreibt, dann gibt es eine Gerade g mit der Eigenschaft, dass die beiden Teilmengen der Kinder sich ergeben, wenn man M durch g in zwei Hälften schneidet. (Die Gerade gehört zu keiner der beiden Hälften.)

Der einer binären planaren Partition zugrunde liegende Baum wird auch „Partitionsbaum“ genannt.

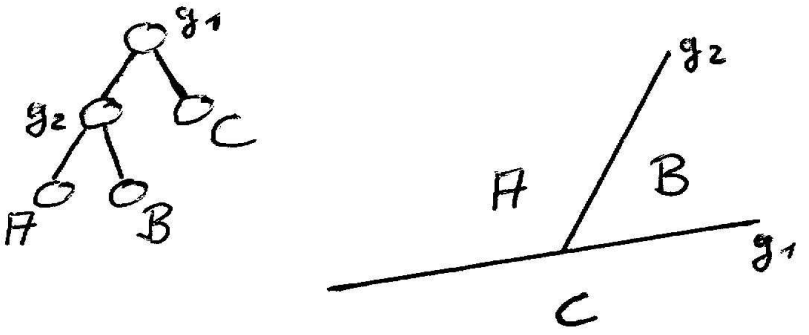
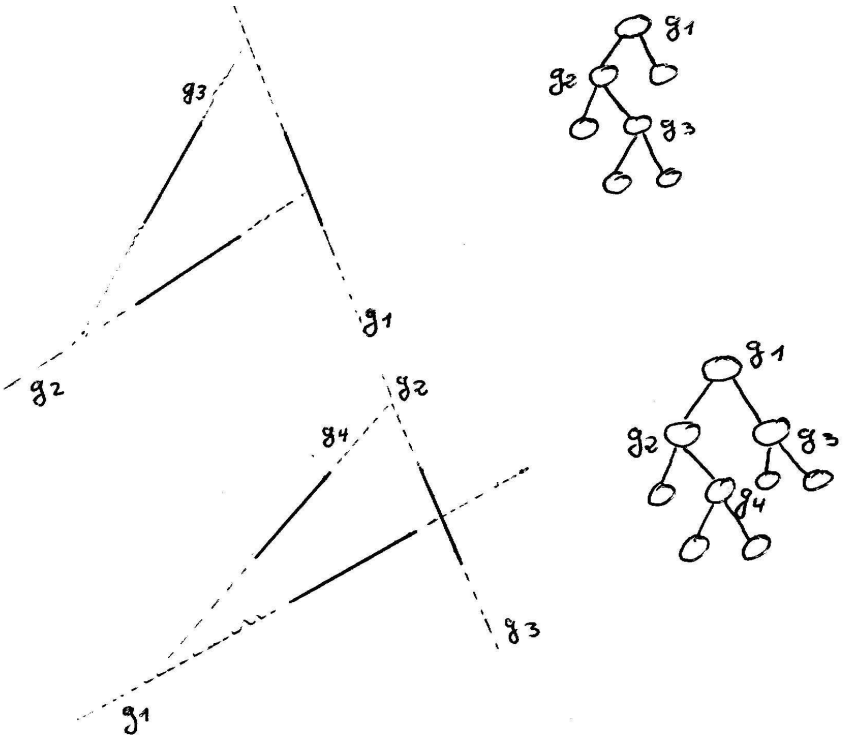


Abbildung 2.2: Links ein Partitionsbaum, rechts die zugehörige Partition.

Die Größe des Partitionsbaums ist definiert als die Zahl der Blätter.
Man überlegt sich leicht, dass ein Partitionsbaum mit ℓ Blättern insgesamt $2 \cdot \ell - 1$ Knoten besitzt, es also $\ell - 1$ innere Knoten gibt (entsprechend $\ell - 1$ verwendeten Trenngeraden). Bei der Größe ist es also relativ egal, ob man die Blätter oder die Anzahl der Trenngeraden zählt.

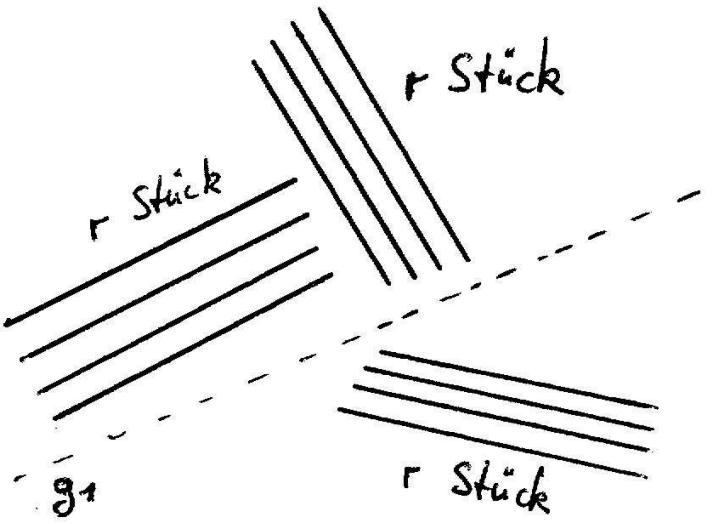
2.2 Definition. Gegeben seien Strecken $s_1, \ldots, s_n \in \mathbf{R}^2$, die sich nicht schneiden. Eine binäre planare Partition heißt *binäre planare Partition für* $s_1, \ldots, s_n$, wenn alle Strecken mit jedem der Teilgebiete in den Blättern einen leeren Schnitt haben. (Salopper: Jede Strecke „verschwindet“ hinter einer der Geraden im Partitionsbaum.)



Wenn Strecken $s_1, \ldots, s_n$ gegeben sind, dann sucht man eine binäre planare Partition für $s_1, \ldots, s_n$ mit möglichst wenig Blättern.

2.3 Definition. Gegeben seien Strecken $s_1, \ldots, s_n \in \mathbf{R}^2$, die sich nicht schneiden. Eine binäre planare Partition für $s_1, \ldots, s_n$ heißt *Autopartition*, falls jede gewählte Gerade durch (mindestens) eine der Strecken s_i, $i \in \{1, \ldots, n\}$, geht.

Autopartitionen sind spezielle Partitionen. Daher könnte es sein, dass es Strecken $s_1, \ldots, s_n$ gibt, für die es eine binäre planare Partition mit „wenigen“ Blättern gibt, bei denen aber jede Autopartition viele Blätter benötigt. Hier bringen wir ein Beispiel, bei dem dieser Effekt andeutungsweise zu erkennen ist.



Diese Lage der $3r$ Strecken führt zu einem Partitionsbaum mit $3r+1$ inneren Knoten. Allerdings kann man zeigen, dass jede Autopartition mindestens $4r$ innere Knoten haben muß.

Bemerkungen: Man weiß nicht, ob es immer eine binäre planare Partition für $s_1, \ldots, s_n$ mit $O(n)$ Blättern gibt. Aber es existiert ein randomisierter Algorithmus, der im Erwartungswert mit $O(n \log n)$ Blättern auskommt. Diesen stellen wir gleich vor.

Wofür sind binäre planare Partitionen überhaupt zu gebrauchen? Wir stellen einen Anwendungsfall vor:

Algorithmus: Der „Painter's Algorithm“ funktioniert wie folgt: Male Objekte, die im Hintergrund sind und übermale sie später. Die Reihenfolge wird durch eine binäre planare Autopartition bestimmt. Zu Beginn wird `Painters`(Wurzel) aufgerufen.

Algorithmus `Painters`(v)

1. `falls` v Blatt ist `return`;
2. `ansonsten` betrachte die zum Knoten gehörende Gerade g;
   - `Painters`(das Kind, das für den Betrachter *hinter* g liegt);
   - male die Strecke(n), die hinter g verschwinden;
   - `Painters`(das andere Kind);

Offensichtlich wird der gesamte Baum traversiert. Damit die Laufzeit recht gering ist, sollte die Anzahl der Knoten möglichst klein sein, also (siehe oben) die Anzahl der Blätter.

2.1.1 Algorithmus RAND-binary-planar-Partition

Input: Strecken $s_1, \ldots, s_n$, die sich nicht schneiden. Dies ist keine große Einschränkung, da wir Strecken, die sich schneiden, in mehrere zerlegen können, die sich nicht schneiden. (Siehe Abbildung 2.3.)

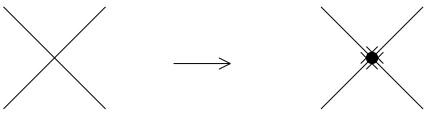


Abbildung 2.3: Zwei Strecken werden in vier sich nicht schneidende Strecken zerlegt.

Output: Eine binäre planare Autopartition für $s_1, \ldots, s_n$, beschrieben durch einen Partitionsbaum.

Starte mit dem Partitionsbaum, der aus einem Knoten besteht.
Wähle gemäß der Gleichverteilung eine der $n!$ Permutationen auf $\{1, \ldots, n\}$, nenne diese Permutation π.

`for` $k = 1$ `to` n
  `do` betrachte die Strecke $s_{\pi(k)}$ und alle Blätter, die mit der Strecke einen nicht-leeren Schnitt haben. Für jedes solche Blatt splitte es auf (zwei Kinder anhängen) und wähle Gerade durch $s_{\pi(k)}$ als Trenngerade.
end;

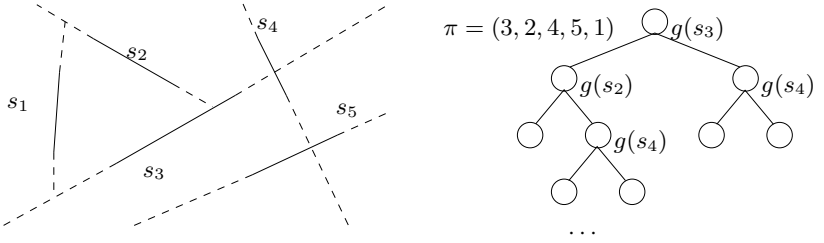
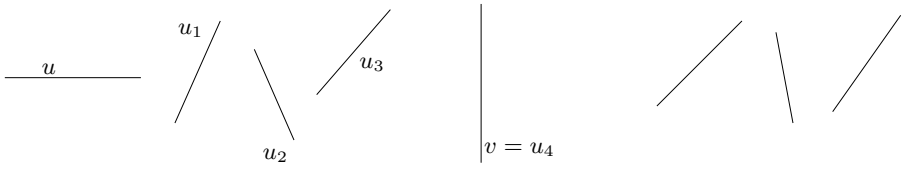


Abbildung 2.4: Beispiel für einen Partitionsbaum.

2.4 Satz. *Der Algorithmus konstruiert einen Partitionsbaum, dessen Blätterzahl im Erwartungswert durch $O(n \log n)$ beschränkt ist.*

Beweis: Für eine Strecke u bezeichne $g(u)$ die Gerade durch u. Betrachte zwei Strecken u und v mit der Eigenschaft, dass $g(u)$ die Strecke v schneidet.



$i(u, v)$ sei die Anzahl der Strecken, die (inklusive v) durch $g(u)$ „auf dem Weg von u zu v“ aufgeschnitten werden. (Damit ist $i(u, v) = 0$, wenn $g(u)$ die Strecke v überhaupt nicht schneidet.) Wir wählen als Indikatorvariable $C_{u,v}$, und zwar wie folgt:

$$C_{u,v} := \begin{cases} 1 & \text{falls im Partitionsbaum die Strecke } v \text{ von der} \\ & \text{Geraden } g(u) \text{ in zwei Stücke zerlegt wird} \\ 0 & \text{sonst} \end{cases}$$

(In Abbildung 2.4 ist $C_{s_3,s_4} = 1$ und $C_{s_2,s_3} = 0$.) Mit dieser Wahl ist die Anzahl der inneren Knoten im Partitionsbaum zu beschreiben als $n + \sum_{u \neq v} C_{u,v}$ und somit ist die *erwartete* Anzahl an Blättern im Partitionsbaum

$$\begin{aligned} & 1 + n + \sum_{u \neq v} \mathbf{E}\left[C_{u,v}\right] \\ = \quad & 1 + n + \sum_{u \neq v} \mathbf{Prob}\left(C_{u,v} = 1\right). \end{aligned}$$

Notwendig für $C_{u,v} = 1$ ist, dass u in der Permutation vor v kommt, aber auch, dass alle anderen u_i nach u in der Permutation kommen, also müssen alle $u_1, u_2, u_3, \ldots, u_{i(u,v)} = v$ in der Permutation nach u kommen. Wie groß ist die Wahrscheinlichkeit dafür?

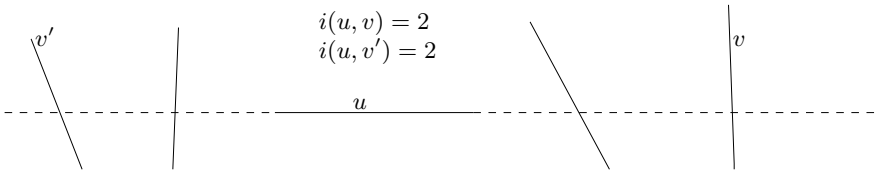
2.5 Lemma. *Gegeben: $S \subseteq \{1, \ldots, n\}$ und ein $x \in S$. Dann gilt: die Wahrscheinlichkeit, dass in einer nach der Gleichverteilung gewählten Permutation auf $\{1, \ldots, n\}$ das Element x das vorderste Element aus S ist, ist $\frac{1}{|S|}$.*

Beweis: Es gibt $|S|!$ viele Permutationen und bei $(|S| - 1)!$ vielen davon steht x vorne.

Aus diesem Lemma folgt: $\mathbf{Prob}\left(C_{u,v} = 1\right) \leq \frac{1}{i(u,v)+1}$.
Das Ungleichheitszeichen ergibt sich, da die obige Bedingung nur eine *notwendige* Bedingung war. Zur Schreibvereinfachung sei $S(u) := \{v \mid \ g(u) \text{ schneidet } v\}$ die Menge aller Strecken, die von $g(u)$ zerschnitten werden. Dann erhalten wir:

$$\mathbf{E}\left[X_n\right] \quad \leq \quad 1 + n + \sum_{u} \sum_{v \in S(u)} \frac{1}{i(u,v) + 1}.$$

Für festes u: wieviele Strecken v mit $i(u,v) = 2$ kann es geben?



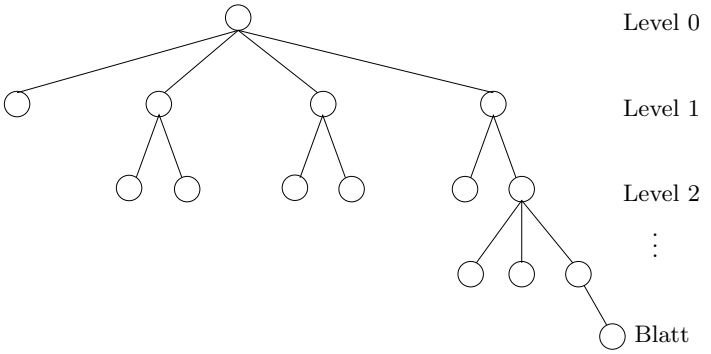
Natürlich maximal zwei. (Einmal eine Strecke rechts von u, einmal eine links von u.)
Allgemein gilt: Bei festem u kann $i(u,\cdot) = j$ *maximal* zweimal auftreten. Damit ergibt sich:

$$\begin{aligned} \mathbf{E}\left[X_n\right] \quad & \leq \quad n + 1 + 2 \cdot \sum_{u} (\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}) \\ & = \quad n + 1 + \sum_{u} 2 \cdot (H_n - 1) \leq 2n \cdot H_n \\ & = \quad O(n \log n). \end{aligned}$$

□

2.2 Spielbaumauswertung

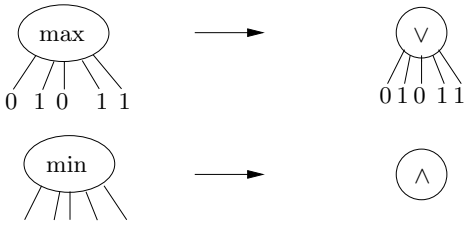
2.6 Definition. Ein Spielbaum ist ein gerichteter Baum mit Wurzel, in dem jedes Blatt v eine Bewertung zahl(v) trägt.



Der Spielbaum soll ausgewertet werden, d.h. berechne wert(wurzel). Dabei ist definiert:

$$\text{wert}(v) = \begin{cases} \text{zahl}(v), & \text{falls } v \text{ Blatt.} \\ \min\{\text{wert}(v_1), \ldots, \text{wert}(v_r)\} & \text{falls } v \text{ die Kinder } v_1, \ldots, v_r \text{ hat} \\ & \text{und } v \text{ auf geradem Level liegt.} \\ \max\{\text{wert}(v_1), \ldots, \text{wert}(v_r)\} & \text{falls } v \text{ die Kinder } v_1, \ldots, v_r \text{ hat} \\ & \text{und } v \text{ auf ungeradem Level liegt.} \end{cases}$$

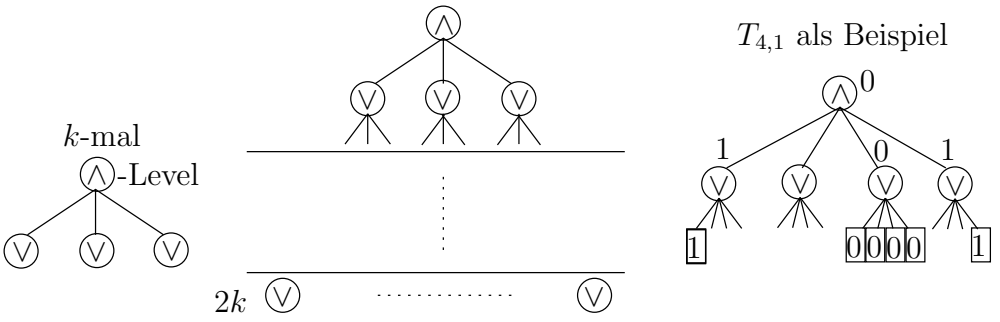
Der Spielbaum stellt ein 2-Personenspiel mit abwechselnder Reihenfolge dar.
Wir betrachten den Spezialfall, dass an den Blättern nur Einsen und Nullen stehen.



→ Spielbaum hat abwechselnd ∧ bzw. ∨-Level

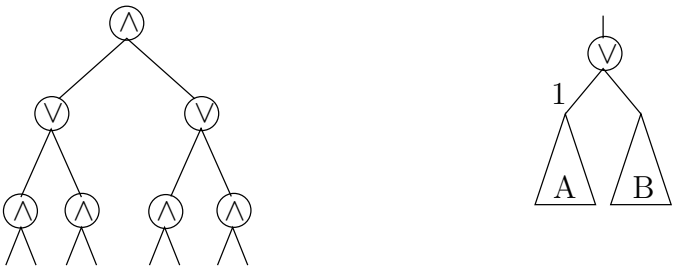
2.7 Definition. $\mathcal{T}_{d,k}$ bezeichne die Menge der Bäume mit den folgenden drei Eigenschaften:

1. Jeder innere Knoten hat genau d Kinder.
2. Alle Blätter haben Entfernung $2k$ von der Wurzel.
3. Für alle Blätter v gilt $\text{zahl}(v) \in \{0, 1\}$.

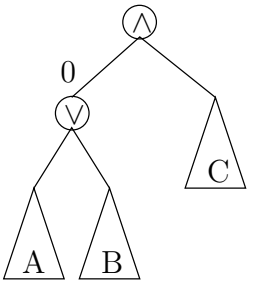


Allgemein:
Input: $x \in \{0,1\}^{d^{2k}}$ (Belegung der Blätter, $N = d^{2k}$ Blätter)
Output: wert(wurzel)
Als Komplexität eines Algorithmus zählen wir die Anzahl der gelesenen Blätter.
Man kann zeigen: Für jeden deterministischen Algorithmus gibt es eine Belegung der Blätter,

so dass der Algorithmus alle Blätter anschauen muss. Also ist die worst-case-Komplexität eines deterministischen Algorithmus immer N.
Wir zeigen: Es existiert ein Las-Vegas-Algorithmus, der auf *jedem* Input erwartete Komplexität $O(N^{0,793})$ hat. Wir betrachten o.B.d.A. den Fall $d = 2$. Wie kann Randomisierung prinzipiell helfen?



Angenommen, das $\vee$-Gatter liefert eine 1 nach oben weiter. Dann liefert einer der beiden Teilbäume A oder B eine 1. Wenn man den auszuwertenden Teilbaum zufällig wählt, und zwar mit Wahrscheinlichkeit $1/2$ jeweils A bzw. B, dann trifft man mit Wahrscheinlichkeit mindestens $1/2$ den 1 liefernden Teilbaum und der andere Teilbaum braucht nicht mehr ausgewertet zu werden.
Angenommen, auf Input x gibt das $\vee$-Gatter 0 nach oben weiter.



Dann müssen A und B ausgewertet werden, aber C braucht nicht ausgewertet zu werden! Wir haben uns jetzt zwar nur $\vee$-Gatter angesehen, aber analoge Aussagen gelten natürlich auch bei $\wedge$-Gattern.

2.2.1 Ein randomisierter Spielbaumauswerter

Algorithmus `RAND-AUSWERT(v:Knoten)`

```
begin
   if v ist Blatt return(zahl(v));     #eine Leseoperation !
   Seien v_1 und v_2 die Kinder von v;
   Wirf eine faire Münze. Falls Kopf: Setze (a := v_1, b := v_2).
   Sonst: (a := v_2, b := v_1).
   temp:=RAND-AUSWERT(a);
   if v ist UND-Knoten und temp=0 then return(0);
   if v ist ODER-Knoten und temp=1 then return(1);
   return(RAND-AUSWERT(b));
end;
```

Wir wollen die erwartete Laufzeit des randomisierten Spielbaumauswertungsalgorithmus analysieren. Diese Laufzeit wird i.W. durch die Anzahl der gelesenen Blätter bestimmt. Daher definieren wir:

2.8 Definition. Für einen Spielbaum T und eine natürliche Zahl $k \geq 0$ sei

$\mathrm{ELO}(T)$:= Erwartete Anzahl an Blättern, die der Algorithmus im Spielbaum T liest.
ELO_k := $\max_k \{\mathrm{ELO}(T) \mid T \text{ ist Baum aus } \mathcal{T}_{2,k}\}$.

Zur Erinnerung: Bäume aus $\mathcal{T}_{2,k}$ haben $N = 4^k$ Blätter.
(ELO steht übrigens für „Erwartete Anzahl an Leseoperationen“.)

2.9 Behauptung. Für alle $k \geq 0$ und $N = 4^k$ gilt: $\forall T \in \mathcal{T}_{2,k} : \mathrm{ELO}(T) \leq N^{0.793}$, also $\mathrm{ELO}_k \leq N^{0.793}$.

Beweis: Durch Induktion zeigen wir, dass $\mathrm{ELO}_k \leq 3^k$ gilt. Die Aussage der Behauptung folgt dann, denn es ist

$$3^k = 2^{k \log 3} = 4^{\frac{k}{2} \log 3} = (4^k)^{\frac{\log 3}{2}} = N^{\frac{\log 3}{2}} \leq N^{0.793}.$$

Induktionsanfang: $k = 0$. Der Spielbaum besteht dann aus 1 Blatt. Daher gilt für alle Bäume T aus $\mathcal{T}_{2,k}$, dass $\mathrm{ELO}(T) = 1 \leq 3^0$ ist.

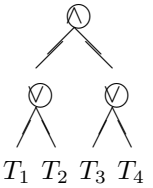
Bevor wir den Induktionsschritt machen, betrachten wir den Fall, dass ein $\vee$-Gatter vorliegt, welches Bäume T_1 und T_2, beide aus $\mathcal{T}_{2,k}$, als Eingabe bekommt und eine 1 als Ausgabe berechnet.

$T =$ 

Mit Wahrscheinlichkeit mindestens (1/2) braucht unser Spielbaumauswerter nur einen der beiden Teilbäume auszuwerten. Dies tritt dann ein, wenn er als erstes einen Teilbaum auswählt, der eine 1 berechnet. O.B.d.A. gehen wir davon aus, dass T_1 eine 1 berechnet. Dann ist

$$\mathrm{ELO}(T) \leq \frac{1}{2}\,\mathrm{ELO}(T_1) + \frac{1}{2}\,(\mathrm{ELO}(T_1) + \mathrm{ELO}(T_2)) \leq (3/2) \cdot \mathrm{ELO}_k.$$

Dies gilt, da T_1 und T_2 nach Annahme aus $\mathcal{T}_{2,k}$ sind.
Nun kommen wir zum Induktionsschritt und betrachten einen Baum T aus $\mathcal{T}_{2,k+1}$.

$T =$ 

Wir haben drei Fälle zu betrachten:

1. Beide $\vee$-Gatter berechnen eine 1: Dann wertet der Algorithmus beide $\vee$-Gatter aus und somit ist $\mathrm{ELO}(T) \leq 1.5\ \mathrm{ELO}_k + 1.5\ \mathrm{ELO}_k = 3\ \mathrm{ELO}_k$.

2. Ein $\vee$-Gatter berechnet eine 0, das andere (o.B.d.A. das linke) berechnet eine 1:
   Mit Wahrscheinlichkeit 1/2 wird das rechte zuerst gewählt, dann werden im Erwartungswert 2 ELO_k (für T_3 und T_4) viele Blätter gelesen. Das linke $\vee$-Gatter wird mit Wahrscheinlichkeit 1/2 zuerst gewählt, dann werden im Erwartungswert höchstens $1.5\ \mathrm{ELO}_k + 2\ \mathrm{ELO}_k = 3.5\ \mathrm{ELO}_k$ viele Blätter gelesen. Wir erhalten insgesamt:

$$\mathrm{ELO}(T) \leq \frac{1}{2}\,(2\ \mathrm{ELO}_k + 3.5\ \mathrm{ELO}_k) < 3\ \mathrm{ELO}_k.$$

3. Beide $\vee$-Gatter liefern eine 0. Dann wird genau ein $\vee$-Gatter ausgewertet und wir erhalten somit $\mathrm{ELO}(T) \leq 2\ \mathrm{ELO}_k$.

In allen drei Fällen haben wir $\mathrm{ELO}(T) \leq 3 \cdot \mathrm{ELO}_k$, also $\mathrm{ELO}_{k+1} \leq 3\ \mathrm{ELO}_k \leq 3^{k+1}$, wobei die letzte Ungleichung aus der Induktionsvoraussetzung folgt. □

Unser Ziel im folgenden wird es sein, zu skizzieren, wie man beweisen kann, dass der obige randomisierte Algorithmus derjenige ist, der (zumindest asymptotisch) für das Problem die beste erwartete Laufzeit hat.
Zu diesem Zweck greifen wir auf Begriffe und Resultate aus der Spieltheorie zurück.

2.10 Definition. Ein *Zwei-Personen-Nullsummenspiel* ist gegeben durch eine Auszahlungsmatrix M sowie zwei Spieler R und C. Spieler R wählt eine Zeile i, Spieler C wählt eine Spalte j. Als Ergebnis des Spiels zahlt Spaltenspieler C den Matrixeintrag $M_{i,j}$ an R.

Aus der Definition folgt direkt, dass Spieler C das Ergebnis des Spiels minimieren möchte, Spieler R es maximieren möchte.

Beispiel:

	Papier	Stein	Schere
Papier	0	1	−1
Stein	−1	0	1
Schere	1	−1	0

-1 = Spaltenspieler gewinnt

Der Begriff „Nullsumme“ im Namen des Spiels erklärt sich daraus, dass die Summe des Gezahlten und Erhaltenen insgesamt 0 ist.
Die Wahl einer Zeile wird (reine) Strategie von Spieler R genannt, die Wahl einer Spalte (reine) Strategie von Spieler C.
Für eine Matrix definieren wir als Zeilenminimum die kleinste in der Zeile stehende Zahl, genauer definieren wir das i-te Zeilenminimum als $a_i := \min_j\{M_{i,j}\}$.
Entsprechend kann man auch Spaltenmaxima definieren. Ein Beispiel einer 3×3-Auszahlungsmatrix, bei der zusätzlich schon oberhalb bzw. links davon die Spaltenmaxima bzw. Zeilenminima eingetragen sind:

	20	18	55
−5	20	10	−5
13	13	18	55
7	10	7	11

Strategie i für Spieler R heißt optimal genau dann, wenn Zeile i das größte Zeilenminimum enthält, wenn also $a_i \geq a_j$ für alle $j \neq i$ ist. Der Wert

$$V_R^{rein} = \max_i \min_j M_{i,j} = \max a_i$$

gibt die größte Auszahlung an, die sich Spieler R *garantieren* kann. Es ist das Maximum aller Zeilenminima. Der analoge Wert für den Spaltenspieler C ist das Minimum aller Spaltenmaxima, also

$$V_C^{rein} = \min_j \max_i M_{i,j}.$$

Beim Stein-Schere-Papier-Spiel ist $V_R^{rein} = -1$ und $V_C^{rein} = 1$.
Wir zeigen nun in folgendem Lemma die Aussage: Für alle Auszahlungsmatrizen gilt: $V_R^{rein} \leq V_C^{rein}$.

2.11 Lemma. *Gegeben sei eine $m \times n$-Matrix mit Zeilenminima $a_1, \ldots, a_m$ und Spaltenmaxima $b_1, \ldots, b_n$. Dann gilt für alle i, j: $a_i \leq b_j$. Also* $\max_i a_i \leq \min_j b_j$.

Beweis: Das Element an der Stelle (i,j) in der Matrix heiße $M_{i,j}$. Es ist nach Definition $a_i \leq M_{i,j}$. Ebenso nach Definition ist $b_j \geq M_{i,j}$. Die Aussage folgt. □

Bei Nullsummenspielen kann auch Ungleichheit vorkommen, wie man bei Stein-Schere-Papier sieht: Es ist $V_R^{rein} = -1$ und $V_C^{rein} = 1$.
Nun betrachten wir so genannte „gemischte Strategien". Gegeben ist eine Auszahlungsmatrix M mit m Zeilen und n Spalten. Gemischte Strategien für die beiden Spieler sind Wahrscheinlichkeitsverteilungen $p = (p_1, \ldots, p_m)$ und $q = (q_1, \ldots, q_n)$. Interpretation:
Spieler R wählt eine Zeile i gemäß der Wahrscheinlichkeitsverteilung $p = (p_1, \ldots, p_m)$, Spieler C wählt eine Spalte j gemäß der Wahrscheinlichkeitsverteilung $q = (q_1, \ldots, q_n)$. Spieler C zahlt dann $M_{i,j}$ an R.
Da das Spiel nicht mehr deterministisch ist, ist die Auszahlung eine Zufallsvariable und diese hat den Erwartungswert

$$\mathbf{E}\,[\text{Auszahlung}] = \sum_{i,j} p_i q_j M_{i,j} = p \cdot M \cdot q^T.$$

Wenn Spieler R die gemischte Strategie p wählt, dann ist ihm der Mindestgewinn $\min_q \; p \cdot M \cdot q^T$ sicher.
Wenn er nun die für sich pessimistische Haltung einnimmt und davon ausgeht, dass sein Gegenüber C immer die für C beste Antwort parat hat, dann sollte R seine gemischte Strategie p so wählen, dass $\min_q \; p \; M \; q^T$ möglichst groß ist. Wir erhalten:
Die Auszahlungen, die beide Spieler sich sichern können, heißen:

$$\begin{aligned} V_R &= \max_p \min_q \; p \; M \; q^T \text{ sowie} \\ V_C &= \min_q \max_p \; p \; M \; q^T. \end{aligned}$$

(Da reine Strategien Spezialfälle von gemischten Strategien sind, gilt logischerweise $V_R^{rein} \leq V_R$ und $V_C \leq V_C^{rein}$.)
Wir zitieren hier ohne Beweis ein berühmtes Resultat aus der Spieltheorie, nämlich:

2.12 Satz (Das Minimaxtheorem von von Neumann)**.** *Für Zwei-Personen-Nullsummenspiele, bei denen gemischte Strategien zugelassen sind, gilt $V_R = V_C$.*

Beim Spiel Stein-Schere-Papier ergibt sich bei Wahl der gemischten Strategien $p = (1/3, 1/3, 1/3), q = (1/3, 1/3, 1/3)$, dass $V_R = 0$ und $V_C = 0$ ist.

Wenn wir mit $e_1^T, \ldots, e_n^T$ die n Einheitsvektoren bezeichnen, dann gilt:

$$\begin{aligned} M \cdot q^T &= M \cdot (q_1 \cdot e_1^T + \cdots + q_n \cdot e_n^T) \\ &= q_1 \cdot M \cdot e_1^T + \cdots + q_n \cdot M \cdot e_n^T. \end{aligned}$$

Also

$$p \cdot M \cdot q^T = q_1 \cdot p \cdot M \cdot e_1^T + \cdots + q_n \cdot p \cdot M \cdot e_n^T.$$

(Man kann also $p \cdot M \cdot q^T$ als konvexe Linearkombination der $p \cdot M \cdot e_i^T$ ansehen.)
Wenn man q beliebig als Wahrscheinlichkeitsverteilung wählen darf, dann wird $p \cdot M \cdot q^T$ also minimal, wenn man $q_i = 1$ setzt für dasjenige i, wo $p \cdot M \cdot e_i^T$ minimal ist und alle anderen q_j auf 0 setzt. Mit dieser Überlegung haben wir auch erhalten:

2.13 Satz (Loomis' Theorem)**.** *Für Zwei-Personen-Nullsummenspiele gilt:*

$$\begin{aligned} V_R &= \max_p \min_j p \cdot M \cdot e_j^T \, (e_j = j\text{-ter Einheitsvektor}). \\ V_C &= \min_q \max_i e_i \cdot M \cdot q^T. \\ V_R &= V_C. \end{aligned}$$

Uminterpretation im Sinne der randomisierten Algorithmen

Ein randomisierter Algorithmus kann in bestimmten Fällen als Wahrscheinlichkeitsverteilung über deterministische Algorithmen gesehen werden. Wir wollen dies an einem Beispiel klarer machen:
Angenommen, ein randomisierter Algorithmus A_{rand} greift bei seinem Ablauf immer auf exakt T Zufallsbits zu und macht davon seine Entscheidungen abhängig.
Wenn wir alle T Zufallsbits festlegen, sagen wir, auf das Muster $r = (r_1, \ldots, r_T)$, dann erhalten wir aus A_{rand} einen deterministischen Algorithmus A_r.
A_{rand} kann nun auch als Wahrscheinlichkeitsverteilung über die (endlich vielen) deterministischen Algorithmen A_r gesehen werden, denn man kann die T Zufallsbits zu Beginn auswürfeln und dann den zugehörigen Algorithmus A_r starten. (Für Experten der Vorlesung GTI: Ein bißchen erinnert die Sichtweise an den Rate-Verifikationsmodus einer nichtdeterministischen Turingmaschine.)
Mit dieser Sichtweise eines randomisierten Algorithmus können wir nun eine Auszahlungsmatrix M definieren. Die Zeilen entsprechen den Inputs, die Spalten den möglichen deterministischen Algorithmen. An Stelle $M_{i,j}$ in der Matrix tragen wir die Laufzeit ein, die der Algorithmus Nummer j auf dem Input Nummer i hat. Bei dieser Sichtweise ist der Spaltenspieler der Algorithmendesigner, der die Laufzeit minimieren möchte und der Zeilenspieler der Gegner, der die Laufzeit maximieren will.
(In unserem Beispiel mit dem Spielbaumauswerter kann man auch die Anzahl der gelesenen Blätter eintragen, da dieses die Zahl ist, die uns interessiert.)
Die Auszahlung kann man nun auch interpretieren:
a) Wenn wir eine Wahrscheinlichkeitsverteilung p über die Inputs haben (im folgenden Inputverteilung genannt), dann gibt die Zahl $\min_j p \cdot M \cdot e_j^T$ die Laufzeit des besten deterministischen Algorithmus für die Inputverteilung p an.
b) Wenn wir einen randomisierten Algorithmus haben, der als Wahrscheinlichkeitsverteilung q über deterministische Algorithmen gegeben ist, dann kann man $\max_i e_i \cdot M \cdot q^T$ interpretieren als worst-case-erwartete Laufzeit des randomisierten Algorithmus. (Das ist schwierig zu formulieren. Gemeint ist: Für jeden Input hat der randomisierte Algorithmus eine erwartete Laufzeit. Die worst-case-erwartete Laufzeit ist das Maximum über alle möglichen Inputs von diesen erwarteten Laufzeiten.)
c) Wenn wir einen randomisierten Algorithmus haben, der gegeben ist durch eine Wahrscheinlichkeitsverteilung q über deterministische Algorithmen, und eine Inputverteilung p, dann ist die Auszahlung $p \cdot M \cdot q^T$ die erwartete Laufzeit des randomisierten Algorithmus auf der gegebenen Inputverteilung.
Wichtig bei der Interpretation: die Anzahl der möglichen deterministischen Algorithmen muss endlich sein.
Man kann nun ahnen, dass das Theorem von Loomis Zusammenhänge zwischen Laufzeiten von randomisierten Algorithmen und Laufzeiten von deterministischen Algorithmen auf Inputverteilungen herstellt. Wir haben zwanglos zu Yaos Minimaxprinzip übergeleitet.

2.3 Yaos Minimaxprinzip

Yaos Minimaxprinzip stellt einen Zusammenhang zwischen deterministischen Algorithmen und randomisierten Algorithmen her.
Für das betrachtete Problem sei sowohl die Menge der Inputs als auch die Menge der deterministischen Algorithmen endlich.
Sei p eine Wahrscheinlichkeitsverteilung auf den Inputs und q eine Wahrscheinlichkeitsverteilung auf den deterministischen Algorithmen. Wir definieren:

- Wenn A ein deterministischer Algorithmus ist und I ein Input, dann sei $L(I, A)$ die Laufzeit des deterministischen Algorithmus A auf Input I.

- $L(p, A)$ bezeichne die erwartete Laufzeit des deterministischen Algorithmus A auf den gemäß der Verteilung p ausgewürfelten Inputs.

- $L(I, q)$ bezeichne die erwartete Laufzeit auf Input I, wenn man den auf I anzuwendenden deterministischen Algorithmus gemäß q auswürfelt. $L(I, q)$ kann als erwartete Laufzeit des (durch q beschriebenen) randomisierten Algorithmus auf Input I angesehen werden. $\max_I L(I, q)$ ist also die erwartete Laufzeit des randomisierten Algorithmus auf einem worst-case-Input.

In der Spieltheorie hatten wir uns Zwei-Personen-Nullsummenspiele angesehen, die durch eine Auszahlungsmatrix M gegeben waren und hatten Loomis' Theorem als die Aussage

$$\max_p \min_j p^T \cdot M \cdot e_j = \min_q \max_i e_i^T \cdot M \cdot q$$

kennengelernt, wobei e_i der i-te Einheitsvektor ist.

Mit der Sichtweise, dass die Zeilen den Inputs entsprechen und die Spalten den deterministischen Algorithmen, ergibt sich aus Loomis' Theorem die folgende Aussage:

$$\max_p \min_{A \text{ det. Algor.}} L(p, A) = \min_q \max_{I \text{ Input}} L(I, q).$$

Wir lesen diese Aussage nun noch anders: Wenn $\max M_1 = \min M_2$ für zwei Mengen M_1 und M_2 gilt, dann ist $m_1 \leq m_2$ für alle $m_1 \in M_1, m_2 \in M_2$. Hier erhalten wir Yaos Minimaxprinzip:

$$\forall p, q : \min_{A \text{ det. Algor.}} L(p, A) \leq \max_{I \text{ Input}} L(I, q).$$

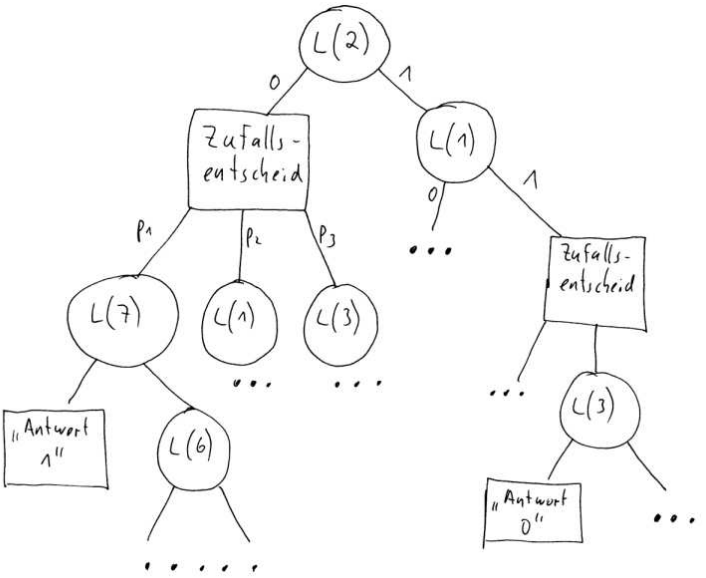
Angenommen, man beschäftigt sich mit einem Problem, für das man nachweisen kann, dass jeder randomisierte Algorithmus A für das Problem sich als Wahrscheinlichkeitsverteilung q_A über deterministische Algorithmen auffassen läßt. Dann folgt aus Yaos Minimaxprinzip:

Die worst-case-erwartete Laufzeit jedes randomisierten Algorithmus A, also $\max_I L(I, q_A)$, ist mindestens so groß wie

$$\max_p \min_{A \text{ det. Algor.}} L(p, A).$$

Wenn es uns also gelingt, eine Wahrscheinlichkeitsverteilung p auf den Inputs zu wählen und nachzuweisen, dass für jeden deterministischen Algorithmus A die erwartete Laufzeit $L(p, A) \geq T$ ist, dann haben wir auch nachgewiesen, dass selbst der beste randomisierte Algorithmus eine worst-case-(erwartete)-Laufzeit mindestens T hat.

Wir betrachten zunächst die Eigenschaft, dass sich im Spielbaumauswertungsszenario jeder randomisierte Algorithmus als Wahrscheinlichkeitsverteilung über deterministische Algorithmen auffassen läßt. Uns interessiert an den Algorithmen nur, welche bzw. wieviele Blätter sie lesen. Daher stellen wir randomisierte Spielbaumauswerter als Entscheidungsbäume dar, in denen in den Knoten entweder ein Blatt i des Spielbaums gelesen werden darf (im folgenden Bild als „$L(i)$“ angedeutet), oder eine Zufallsentscheidung getroffen werden darf. In den Blättern des Spielbaumauswerters steht die gegebene Antwort, die den Wert des ausgewerteten Spielbaums angibt. Ein Beispiel für die Darstellung eines randomisierten Spielbaumauswerters als Entscheidungsbaum:



In *deterministischen* Spielbaumauswertern gibt es keine Knoten mit Zufallsentscheidungen.
Nun ist es ziemlich offensichtlich, wie man einen randomisierten Spielbaumauswerter als Wahrscheinlichkeitsverteilung über deterministische Spielbaumauswerter bekommen kann, wir wollen dies hier nur skizzieren:
Wenn man für jeden Zufallsentscheid nacheinander jeden möglichen Ausgang des Zufallsentscheids durchgeht, erhält man eine Menge von deterministischen Spielbaumauswertern. Man kann auch (durch Multiplizieren der entsprechenden Wahrscheinlichkeiten) die Wahrscheinlichkeit ausrechnen, dass die Zufallsentscheide wie gewählt getroffen werden. Diese gibt dann die Wahrscheinlichkeit an, mit der der deterministische Spielbaumauswerter gewählt wird.
Damit kann man auf die Spielbaumauswerter Yaos Minimaxprinzip anwenden.
Wenn es uns also gelingt, eine Wahrscheinlichkeitsverteilung p auf den Inputs zu wählen und nachzuweisen, dass jeder deterministische Spielbaumauswerter bei diesen Inputs Laufzeit mindestens T hat, so gilt auch für den besten randomisierten Spielbaumauswerter, dass dieser eine worst-case-erwartete-Laufzeit mindestens T hat.
Wir wählen nun eine solche Wahrscheinlichkeitsverteilung. Um die Spielbäume besser handhaben zu können, beobachten wir zunächst, dass man im Spielbaum alle UND- und ODER-Gatter durch NOR-Gatter ersetzen kann ohne die berechnete Funktion zu verändern:
Es ist $NOR(x,y) = \overline{x \vee y} = \overline{x} \wedge \overline{y}$, also

$$NOR(NOR(x_1,x_2),NOR(x_3,x_4)) = \overline{NOR}(x_1,x_2) \wedge \overline{NOR}(x_3,x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4).$$

Damit kann man jeweils ein Paar aus UND-Ebene und ODER-Ebene durch zwei Ebenen aus NOR-Gattern ersetzen. Wir können uns also darauf beschränken, NOR-Spielbäume zu betrachten.
Wir wählen folgende Wahrscheinlichkeitsverteilung auf den Inputs (=Spielbaumblättern) $x_1,\ldots,x_n$: Setze die x_i unabhängig voneinander jeweils mit Wahrscheinlichkeit $p^* := \frac{3-\sqrt{5}}{2} \approx 0.382$ auf 1, sonst auf 0.
Man kann induktiv zeigen, dass jeder Knoten im NOR-Spielbaum mit Wahrscheinlichkeit p^* eine 1 berechnet: Wenn x und y jeweils mit Wahrscheinlichkeit p^* den Wert 1 annehmen, dann liefert NOR(x,y) mit Wahrscheinlichkeit $(1-p^*)^2$ den Wert 1. Es gilt

$$(1-p^*)^2 = (\frac{2}{2} - \frac{3-\sqrt{5}}{2})^2 = (\frac{\sqrt{5}-1}{2})^2 = \frac{1}{4}\cdot(5-2\sqrt{5}+1) = \frac{3-\sqrt{5}}{2} = p^*.$$

Nun ist es unsere Aufgabe, uns anzusehen, wie deterministische Spielbaumauswerter mit dieser Inputverteilung umgehen. Eigentlich müssen wir uns alle deterministischen Spielbaumauswerter ansehen, aber wir können auf eine Aussage zurückgreifen, die im Artikel

Tarsi: Optimal search on some game trees: JACM, 1983, Vol. 30 (No. 3), 389–396

zu finden ist:

Definition: Der Algorithmus „depth-first-pruning“ wertet den Spielbaum so aus, dass er in jedem Knoten zunächst den Wert des linken Teilbaums bestimmt und nur dann den zweiten Teilbaum auswertet, wenn es noch nötig ist. Den Teil „Depth-First“ seines Namens verdankt der Algorithmus der Reihenfolge seines Vorgehens, den Teil „Pruning“ verdankt er der Eigenschaft, dass manchmal der zweite Teilbaum abgeschnitten („pruning“= „abschneiden“) wird.

Proposition: Sei T ein NOR-Spielbaum, in dem jedes Blatt mit Wahrscheinlichkeit q auf 1 gesetzt wird. Der Algorithmus depth-first-pruning hat von allen deterministischen Algorithmen für diese Inputverteilung die beste erwartete Laufzeit.

Anstatt uns alle deterministischen Spielbaumauswerter ansehen zu müssen, brauchen wir also nun nur noch die erwartete Laufzeit von depth-first-pruning zu analysieren. Dies geht relativ einfach.

Wir teilen den Spielbaum in Ebenen ein, wobei Ebene 0 die Blätter des Spielbaums enthält, Ebene 1 die Eltern der Blätter, etc. Die erwartete Laufzeit von depth-first-pruning, um einen Knoten auf Ebene h auszuwerten, sei $W(h)$. Dann ist $W(0) = 1$, da nur das entsprechende Blatt gelesen wird und dann der Output feststeht und induktiv ist

$$W(h) = W(h-1) + (1-p^*) \cdot W(h-1) = (2-p^*) \cdot W(h-1),$$

denn depth-first-pruning wertet zunächst den linken Teilbaum aus und nur wenn dieser den Wert 0 liefert (was mit Wahrscheinlichkeit $(1-p^*)$ passiert), ist auch der rechte Teilbaum noch auszuwerten. Wir lösen die Rekursionsgleichung und erhalten:

$$W(h) = (2-p^*)^h.$$

Wenn der Spielbaum n Blätter hat, dann liegt die Wurzel des Spielbaums auf Ebene $\log n$ und es ist

$$W(\log n) = (2-p^*)^{\log n} = n^{\log(2-p^*)} \geq n^{0.6942}.$$

Damit haben wir gezeigt, dass depth-first-pruning auf NOR-Spielbäumen mit n Blättern die erwartete Laufzeit von mindestens $n^{0.6942}$ hat, wenn die oben beschriebene Wahrscheinlichkeitsverteilung auf den Inputs gewählt wird.

Mit Yaos Minimaxprinzip folgt nun, dass es für jeden randomisierten Spielbaumauswerter einen Input gibt, auf dem er im Erwartungswert mindestens $n^{0.6942}$ Blätter liest.

Man kann, allerdings mit größerem technischen Aufwand, durch geschicktere Wahl der Wahrscheinlichkeitsverteilung nachweisen, dass in der Tat der von uns in 2.2.1 beschriebene randomisierte Spielbaumauswerter (asymptotisch) optimale Laufzeit hat. Dies wollen wir aber nicht vorführen, da es uns um die wesentlichen Eigenschaften von Yaos Minimaxprinzip ging.

2.4 MaxCut und MinCut, Randomisierte Kontraktion

Das Problem MaxCut ist wie folgt definiert.

Problem MaxCut

Gegeben: Ein Graph $G = (V, E)$ mit Kantengewichten $w : E \to \mathbf{R}$.

Gesucht: Eine Zerlegung der Knotenmenge V des Graphen $G = (V, E)$ in zwei nichtleere Mengen V_1 und V_2 mit $V = V_1 \cup V_2 \quad$ und $\quad V_1 \cap V_2 = \emptyset$, so dass das Gesamtgewicht aller Kanten zwischen diesen beiden Mengen maximal ist.

Sprechweise: Bei einer Zerlegung $V_1 \cup V_2$ bezeichnet man die Menge der Kanten, die zwischen V_1 und V_2 verlaufen, als „Schnitt“.
Beim MinCut-Problem sucht man nach dem *minimalen Gesamtgewicht* aller Kanten. (Und nur für dieses Problem braucht man die Bedingung, dass V_1 und V_2 nichtleer sein sollen.)
Das MinCut-Problem ist in Polynomialzeit lösbar, das MaxCut-Problem hingegen ist NP-hart, d.h. unter der Annahme P $\neq$ NP ist es nicht in Polynomialzeit lösbar. Zur exakten Lösung des MaxCut-Problems sind also keine effizienten Verfahren zu erwarten.
Sehen wir uns einen einfachen Algorithmus für das MaxCut-Problem an:

Algorithmus `RandomCut`
for $i := 1$ **to** n **do**
begin
  Setze $x_i := 1$ bzw. $x_i := 0$ mit Wahrscheinlichkeit $1/2$.
end
Sei $V_1 := \{v_i \in V \mid x_i = 1\}$ und $V_2 := \{v_i \in V \mid x_i = 0\}$.

Wir würfeln also für alle Knoten unabhängig voneinander aus, ob wir sie in die Menge V_1 oder in die Menge V_2 einfügen.

2.14 Satz. *Der Algorithmus liefert in Zeit $O(n)$ eine Zerlegung $V = V_1 \cup V_2$. Für die Zufallsvariable X, die das Gesamtgewicht der Kanten zwischen V_1 und V_2 angibt, gilt:*

$$\mathbf{E}[X] = \frac{w(E)}{2}.$$

$w(E)$ steht dabei abkürzend für das Gewicht $\sum_{e \in E} w(e)$.

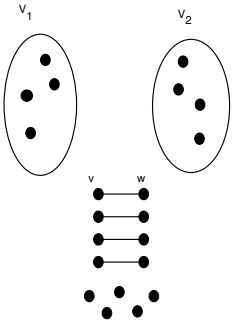
Beweis: Die Aussage über die Laufzeit ist trivial. Für eine beliebige Kante $e = \{v, w\} \in E$ ist die Wahrscheinlichkeit, dass sie zwischen den beiden Mengen verläuft, genau $1/2$. Es muss nämlich entweder v in die Menge V_1 und w in die Menge V_2 gewürfelt werden oder v in die Menge V_2 und w in die Menge V_1. Beide Ereignisse treten jeweils mit Wahrscheinlichkeit $1/4$ ein.
Sei X_e die (Indikator-) Zufallsvariable, die 1 ist, wenn die Kante e zwischen den beiden Mengen verläuft und 0 sonst. Es gilt also $\mathbf{E}[X_e] = 1/2$ für jede Kante $e \in E$. Das Gesamtgewicht der Kanten zwischen V_1 und V_2 ist $X = \sum_{e \in E} X_e \cdot w(e)$, wegen der Linearität des Erwartungswertes ist also $\mathbf{E}[X] = \frac{w(E)}{2}$. □

Man kann das Resultat sogar noch ein wenig verbessern:

2.15 Lemma. *Sei $G = (V, E)$ ein ungerichteter Graph mit Kantengewichten $w : E \to \mathbf{R}$. Sei ein Matching M in G gegeben. Dann kann man in Zeit $O(n)$ eine Zerlegung $V = V_1 \cup V_2$ berechnen, die ein erwartetes Gesamtgewicht von mindestens*

$$\frac{w(E) + w(M)}{2} \text{ hat.}$$

Beweis:



Alle Knoten, die nicht am Matching beteiligt sind, werden (wie bisher) mit Wahrscheinlichkeit jeweils $1/2$ in die Menge V_1 bzw. die Menge V_2 aufgenommen. Für die Knoten, die am Matching beteiligt sind, geht man anders vor:
Sei (v, w) eine Kante des Matchings M. Mit Wahrscheinlichkeit $1/2$ jeweils macht man entweder a) oder b):
a) Füge v zu V_1 und gleichzeitig w zu V_2 hinzu.
b) Füge v zu V_2 und gleichzeitig w zu V_1 hinzu.

Die Analyse sieht nun wie folgt aus: Für eine Kante, die nicht in M liegt, ist die Wahrscheinlichkeit nach wie vor $1/2$, zwischen V_1 und V_2 zu verlaufen. Eine Kante, die in M liegt, verläuft aber auf jeden Fall zwischen V_1 und V_2. Wir erhalten als erwartetes Gesamtgewicht:

$$\frac{w(E \setminus M)}{2} + w(M) = \frac{w(E) + w(M)}{2}.$$

□

2.4.1 MinCut

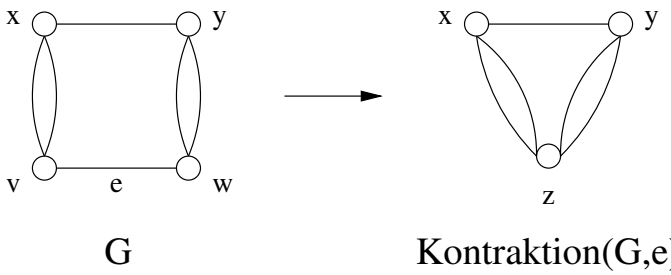
Für unseren Algorithmus benötigen wir eine Operation „Kontraktion“, die zwei gegebene Knoten v und w „zusammenzieht“. Wir beschreiben, wie man zwei gegebene Knoten v und w kontrahieren kann. Falls es eine Kante zwischen v und w gibt, entfernt man diese zunächst.

Eingabe:	Ein Multigraph $G = (V, E)$ und zwei Knoten v und w.
Ausgabe:	Ein Multigraph $G' = (V', E')$ mit $\|V'\| = \|V\| - 1$, wobei G' aus G wie folgt entsteht:

- Für die Kante (Multikante) $e = \{v, w\} \in E$ ersetze die Knoten v und w durch einen neuen (Meta-) Knoten z. Alle anderen Knoten bleiben unverändert.
- Für Knoten $x \neq v, w$ ersetze jede Kante $\{x, v\} \in E$ und $\{x, w\} \in E$ durch die Kante $\{x, z\}$. Die anderen Kanten bleiben erhalten. Sei E' die resultierende Kantenmenge.

Das Resultat der Operation ‚Kontraktion‘ wird, wenn $e = \{v, w\}$ ist, mit Kontraktion(G, e) bezeichnet. Der Graph G' enthält genau einen Knoten weniger als G.

Beispiel:



Die Zeit für die Durchführung der Kontraktion ist offenbar $O(n)$ für $|V| = n$, da jeder Knoten in G maximal $O(n)$ Nachbarn hat. Hierbei ist es für spätere Zwecke von Vorteil, sich zu merken, welche Knoten zu einem Metaknoten gehören.

Algorithmus „Randomisierte Kontraktion“

Wir stellen nun den randomisierten Algorithmus, basierend auf Kontraktionen, vor. Wenn der Eingabegraph G nicht zusammenhängend ist, dann können wir V_1 als die Menge der Knoten einer beliebigen Zusammenhangskomponente von G wählen und V_2 als alle anderen Knoten und die Partition $V_1 \dot\cup V_2$ ist dann minimal. Im folgenden sei also der Eingabegraph zusammenhängend. Durch die Kontraktionen bleibt der Zusammenhang erhalten.

Eingabe: Ein zusammenhängender Multigraph $G = (V, E)$ mit $|V| \geq 2$.
Ausgabe: Eine Partition $V = V_1 \dot\cup V_2$ bzw. die entsprechende Menge der Kanten $e \in E$ zwischen V_1 und V_2.

$H := G$
while H hat mehr als 2 Knoten **do**
  Wähle zufällig gemäß Gleichverteilung eine Kante e aus H.
  $H :=$ Kontraktion(H, e)
end of while
V_1 und V_2 sind die beiden Knotenmengen, die den zwei (Meta-) Knoten in H entsprechen.

Liegen gewichtete Kanten vor (Mehrfachkanten), so werden diese dann natürlich mit einer Wahrscheinlichkeit entsprechend ihrem Gewicht gezogen. Da der Eingabegraph G und somit auch H zusammenhängend ist, ist die Menge der Kanten, aus der wir eine Kante auswählen, nie leer während des Algorithmus.

2.16 Satz. *Für Multigraphen mit n Knoten kann der Algorithmus „Randomisierte Kontraktion“ so implementiert werden, dass er in Zeit $O(n^2)$ ausgeführt werden kann.*

Beweis: Es reichen folgende Beobachtungen:

- Es werden maximal $n - 2$ Kontraktionen durchgeführt.
- Die Zeit für eine Kontraktion ist $O(n)$ für $|V| = n$.
- Also ist die Gesamtzeit $O(n^2)$.

Der Algorithmus ‚Randomisierte Kontraktion‘ terminiert also nach $O(n^2)$ Schritten und liefert eine Partition $V = V_1 \dot\cup V_2$, die aber **nicht** notwendigerweise einen optimalen Cut (Zerlegung) darstellen muss. □

Es ist hier nur noch zu überlegen, wie die zufällige Auswahl einer Kante in Zeit $O(n)$ bewerkstelligt werden kann. Dies sei als Übung überlassen.
Im folgenden bezeichnen wir mit mincut(G) die Anzahl der Kanten im minimalen Schnitt.

2.17 Lemma. *Es gelten folgende drei Aussagen für Graphen $G = (V, E)$ mit $|V| = n$:*

1. *Eine Partition $V = V_1 \dot\cup V_2$ ist genau dann Resultat des Algorithmus ‚Randomisierte Kontraktion‘, wenn im Verlauf des Algorithmus keine Kante kontrahiert wurde, die zwischen V_1 und V_2 verläuft.*
2. *Ist mincut$(G) = k$, dann gilt $|E| \geq k \cdot \frac{n}{2}$.*
3. *Für jede Kante $e \in E$ gilt: mincut$(G) \leq$ mincut(Kontraktion(G, e)).*

Beweis: [zu 2.] Jeder Knoten hat Grad mindestens k, da ansonsten der minimale Schnitt kleiner als k wäre. Also ergibt sich:

$$\sum_{v \in V} \text{grad}(v) \geq k \cdot n$$

und folglich gibt es in G mindestens $\frac{k \cdot n}{2}$ viele Kanten. □

Beweis: [zu 3.] Wenn man einen Metaknoten in einer Menge wieder aufbricht, verändert sich der Schnitt offensichtlich nicht. Das „$\leq$“ ergibt sich daraus, dass man einen „Teil“ des Metaknotens in die andere Menge packen könnte und so einen eventuell kleineren Schnitt bekommen könnte. □

2.18 Satz. *Der Algorithmus „Randomisierte Kontraktion“ liefert mit Wahrscheinlichkeit von mindestens $2/n^2$ einen minimalen Schnitt. Genauer: Sei K die Kantenmenge eines minimalen Schnittes. Die Wahrscheinlichkeit, dass K als Kantenmenge ausgegeben wird, ist mindestens $2/n^2$.*

(Übrigens folgt aus dem Satz als Nebenprodukt, dass es höchstens $n^2/2$ viele minimale Schnitte gibt.)

Beweis: Vor der i-ten Runde liegen $n_i = n-i+1$ Knoten vor. Der Algorithmus liefert die Menge K als Ausgabe genau dann, wenn keine Kante $e \in K$ kontrahiert wird. Sei E_i das Ereignis „es wurde in Runde i keine Kante aus K kontrahiert“.
Sei G_i der Graph, der in der i-ten Runde vorliegt, und sei $|K| = k$. Da durch Kontraktionen der Mincut höchstens größer wird, gilt stets $\text{mincut}(G_i) \geq k$, also ist die Anzahl der Kanten in G_i mindestens $\frac{k \cdot (n-i+1)}{2}$. Höchstens k Kanten in G_i sind aus K, also ist die Wahrscheinlichkeit, dass eine Kante aus K zur Kontraktion ausgewählt wird, durch $\frac{2}{(n-i+1)}$ beschränkt, also:

$$\mathbf{Prob}\left(E_i \mid E_1 \cap \cdots \cap E_{i-1}\right) \geq 1 - \frac{2}{n-i+1}.$$

Somit ergibt sich:

$$\begin{aligned}
\mathbf{Prob}\left(\text{Algorithmus liefert } K\right) &= \mathbf{Prob}\left(E_1 \cap E_2 \cap \cdots \cap E_{n-2}\right) \\
&\geq (1-\frac{2}{n}) \cdot (1-\frac{2}{n-1}) \cdots (1-\frac{2}{3}) \\
&= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{1}{3} \\
&= \frac{1 \cdot 2 \cdots (n-2)}{3 \cdots n} \\
&= \frac{2}{(n-1) \cdot n} \geq \frac{2}{n^2}.
\end{aligned}$$

Damit haben wir den Satz bewiesen. □

Also ist die erwartete Anzahl an Aufrufen, bis ein minimaler Schnitt gefunden wird, höchstens $n^2/2$. Dies bringt aber insgesamt eine schlechte Laufzeit von $O(n^4)$. Wird der Algorithmus „Randomisierte Kontraktion“ N-mal ausgeführt, ist die Wahrscheinlichkeit, jedesmal einen Schnitt auszugeben, der *nicht* minimal ist, höchstens

$$\left(1 - \frac{2}{n^2}\right)^N \leq e^{-2 \cdot \frac{N}{n^2}}$$

Die Erfolgswahrscheinlichkeit ist somit mindestens $1 - e^{-2 \cdot \frac{N}{n^2}}$. Wählt man $N = c \cdot n^2 \cdot \ln n$, so beträgt die Wahrscheinlichkeit mindestens

$$1 - e^{-2c \cdot \ln n} = 1 - n^{-2c}.$$

Wenn der Algorithmus „Randomisierte Kontraktion“ schon stoppt, wenn nur noch t Knoten vorliegen, dann ist die Wahrscheinlichkeit, dass bis dahin keine Kante aus K kontrahiert worden

ist, mindestens $\frac{t\cdot(t-1)}{n\cdot(n-1)}$. Dies bringt uns zu folgendem Algorithmus.

Algorithmus `FastCut`$(G=(V,E))$

1. $n = |V|$.
2. IF $n \leq 6$ THEN berechne min. Schnitt nach irgendeiner trivialen Methode.
3. $t = \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$.
4. Rand. Kontraktion bis auf t Knoten von $G \to H_1$.
   Rand. Kontraktion bis auf t Knoten von $G \to H_2$.
5. `FastCut`(H_1).
   `FastCut`(H_2).
   Gib die kleinere der beiden berechneten Kantenmengen aus.

2.19 Satz. `FastCut` *hat Laufzeit* $O(n^2 \log n)$ *und liefert einen minimalen Schnitt mit Wahrscheinlichkeit mindestens* $\Omega(\frac{1}{\log n})$.

Analyse der Laufzeit „O.B.d.A.-Annahmen“:

1. Beide Kontraktionen in Schritt 4 kosten zusammen Laufzeit *genau* n^2.
2. $T(6) = 1$.
3. $T(n)$ sei die Laufzeit von FastCut auf Graphen der Größe n.

Es gilt also $T(6) = 1$ sowie $T(n) = 2 \cdot T\left(\left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil\right) + n^2$. Es sei $n_0 := n$ und $n_{i+1} := \left\lceil 1 + \frac{n_i}{\sqrt{2}} \right\rceil$. Wenn $n_0 = n \geq 6$ ist, so gibt es stets ein r mit $n_r = 6$. Dies zeigen wir nachher. Wir nehmen zunächst einmal an, dass wir das r kennen und schätzen die Laufzeit in Abhängigkeit von r ab. Sei also nun $n_r = 6$. Es ergibt sich die Rekursionsgleichung:

$$\begin{aligned} T(n) = T(n_0) &= 2 \cdot T(n_1) + n^2 \\ &= 2(2 \cdot T(n_2) + n_1^2) + n_0^2 \\ &= 4 \cdot T(n_2) + 2n_1^2 + n_0^2 \\ &\cdots \\ &= 2^r \cdot T(n_r) + \sum_{i=0}^{r-1} 2^i \cdot n_i^2 \\ &= 2^r + \sum_{i=0}^{r-1} 2^i \cdot n_i^2. \end{aligned}$$

Nun benötigt man die folgende Ungleichung:

$$n_{i+1}^2 \geq \left(\frac{n_i}{\sqrt{2}}\right)^2 = \frac{n_i^2}{2},$$

also $n_i^2 \leq 2 \cdot n_{i+1}^2$, also (induktiv) $n_i^2 \leq 2^{r-i} \cdot n_r^2$ bzw. $2^i \cdot n_i^2 \leq 2^r \cdot n_r^2$. Wir können nun obigen

Term abschätzen durch

$$\begin{aligned} T(n) &= 2^r + \sum_{i=0}^{r-1} 2^i \cdot n_i^2 \\ &\leq 2^r + \sum_{i=0}^{r-1} 2^r \cdot n_r^2 \\ &= 2^r + r \cdot 2^r \cdot n_r^2 \\ &= 2^r + r \cdot 2^r \cdot 36 = O(r \cdot 2^r). \end{aligned}$$

Es bleibt die Aufgabe, r zu berechnen. Da die Funktion $f(x) := \left\lceil 1 + \frac{x}{\sqrt{2}} \right\rceil$ monoton und $f(6) = 6$ ist, ist für $n_0 = n \geq 6$ keines der n_i kleiner als 6. Das folgende Lemma hilft uns nun, das r abzuschätzen:

2.20 Lemma. *Sei $q := \frac{1}{\sqrt{2}}$. Dann gilt für alle $i \geq 0$, dass $n_i \leq \frac{2}{1-q} + n \cdot q^i$ ist. (Dabei ist $\frac{2}{1-q} \approx 6.8284$.)*

Beweis: Durch Induktion. Induktionsanfang: $n_0 = n \leq n + 6$. Induktionsschritt:

$$\begin{aligned} n_{i+1} = \lceil 1 + q \cdot n_i \rceil &\leq 2 + q \cdot n_i \\ &\leq 2 + q \left(\frac{2}{1-q} + n \cdot q^i \right) \\ &= 2 + \frac{2q}{1-q} + n \cdot q^{i+1} \\ &= \frac{2 - 2q + 2q}{1-q} + n \cdot q^{i+1} = \frac{2}{1-q} + n \cdot q^{i+1}. \end{aligned}$$

□

Wählt man $i \geq 2 \cdot \log n + 6$, so folgt $n_i < 7$, also wegen der Ganzzahligkeit $n_i = 6$.
Somit ist $r \leq \lceil 2 \cdot \log n + 6 \rceil$. Setzt man diese Schranke für r in die oben berechnete Schranke für $T(n)$ ein, so ergibt sich eine Laufzeit von $O(n^2 \log n)$. Kommen wir nun zur Analyse der Erfolgswahrscheinlichkeit. Sei E_1 das Ereignis, dass die folgenden beiden Dinge passieren:

1. Bei der Kontraktion zu H_1 wird keine Kante aus K kontrahiert.
2. FastCut liefert auf H_1 einen minimalen Schnitt.

Entsprechend definieren wir E_2 über H_2. FastCut ist erfolgreich, wenn E_1 oder E_2 eintritt.
In H_1 sind mit Wahrscheinlichkeit mindestens $\frac{t(t-1)}{n(n-1)}$ alle Kanten eines minimalen Schnitts K vorhanden. Für $t = \lceil 1 + \frac{n}{\sqrt{2}} \rceil$ ergibt das eine Wahrscheinlichkeit von mindestens $\frac{\frac{n}{\sqrt{2}} \cdot \frac{n}{\sqrt{2}}}{n^2} = \frac{1}{2}$. Damit tritt „1.“ mit Wahrscheinlichkeit mindestens $1/2$ ein.

$$\mathbf{Prob}\left(E_1 \cup E_2\right) = \mathbf{Prob}\left(E_1\right) + \mathbf{Prob}\left(E_2\right) - \mathbf{Prob}\left(E_1 \cap E_2\right).$$

Dies ist monoton in $\mathbf{Prob}\left(E_1\right)$ und $\mathbf{Prob}\left(E_2\right)$, wenn also beide mindestens den Wert p haben, dann hat FastCut Erfolgswahrscheinlichkeit mindestens $2p - p^2$.
Wir betrachten den Rekursionsbaum von FastCut. Die Struktur des Baums ist auf allen Eingaben gleich. Wir definieren: Sei $p(r)$ die kleinste Erfolgswahrscheinlichkeit, über alle Eingaben betrachtet, die zu einem Rekursionsbaum der Tiefe r führen. Wir erhalten mit den obigen Überlegungen bei einem Rekursionsbaum der Tiefe $r + 1$: $\mathbf{Prob}\left(E_1\right) \geq (1/2) \cdot p(r)$, ebenso für $\mathbf{Prob}\left(E_2\right)$ und somit mit der Wahl $p := (1/2) \cdot p(r)$:

$$p(r+1) \quad \geq \quad p(r) - \frac{p^2(r)}{4}.$$

2.21 Satz. *Falls $p(i) \geq \frac{1}{d}$ für irgendein d ist, dann gilt $p(i+1) \geq \frac{1}{d+1}$.*

Beweis: Wir definieren die Funktion $f(x) := x - x^2/4$. Unsere Überlegungen gerade haben gezeigt, dass $p(i+1) \geq f(p(i))$ gilt. Die Funktion f ist eine nach unten offene Parabel mit Scheitel bei $x = 2$. Damit ist sie im Intervall $[0, 1]$ monoton steigend und wir erhalten:

$$p(i) \geq \frac{1}{d} \Rightarrow p(i+1) \geq f\left(\frac{1}{d}\right) = \frac{1}{d} - \frac{1}{4d^2}.$$

Nun muss gelten:

$$\frac{1}{d} - \frac{1}{4d^2} \stackrel{!}{\geq} \frac{1}{d+1} \Leftrightarrow d + 1 - \frac{d+1}{4d} \geq d \Leftrightarrow \frac{d+1}{4d} \leq 1 \Leftrightarrow d \geq \frac{1}{3}.$$

Andererseits ist sogar $d \geq 1$ garantiert, da $p(i)$ eine Wahrscheinlichkeit ist (somit höchstens 1) und somit aus $p(i) \geq 1/d$ folgt, dass $d \geq 1$ ist. □

Da $p(0) = 1 \geq 1/1$, können wir dieses Lemma induktiv verwenden, um zu folgern, dass $p(1) \geq \frac{1}{2}, \ldots, p(i) \geq \frac{1}{i+1}$ ist. Für $r \leq 2 \log n + 7$ ergibt sich eine Erfolgswahrscheinlichkeit von $p(r) = \Omega\left(\frac{1}{\log n}\right)$.

Fazit zum Algorithmus `FastCut`

- `FastCut` hat Laufzeit $O(n^2 \log n)$.
- Die Wahrscheinlichkeit, *einen* minimalen Schnitt auszugeben, ist $\Omega(\frac{1}{\log n})$.

Eine Randbemerkung scheint angebracht: Bei dem Algorithmus „Randomisierte Kontraktion“ galt sogar: Für alle minimalen Schnitte K ist $\mathbf{Prob}\,(K \text{ wird ausgegeben}) = \Omega(1/n^2)$. Diese analoge Eigenschaft (mit dem Term $\Omega(1/\log n)$ statt $\Omega(1/n^2)$) gilt bei `FastCut` nicht mehr. Wir betrachten als Gegenbeispiel den vollständigen Graphen auf n Knoten, K_n. Die minimalen Schnitte sind für beliebiges $v \in V$ von der Form $V_1 = \{v\}$ und $V_2 = V \setminus \{v\}$. Damit haben wir n verschiedene minimale Schnitte und es *kann* nicht gelten, dass jeder davon mit Wahrscheinlichkeit $\Omega(1/\log n)$ ausgegeben wird.

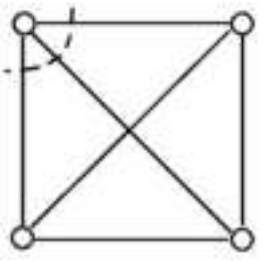


Abbildung 2.5: Ein minimaler Schnitt für den K_4.

Um nun zu einer annehmbaren Erfolgswahrscheinlichkeit zu kommen, iterieren wir den Algorithmus `FastCut`. Die Wahrscheinlichkeit, einen minimalen Schnitt auszugeben, ist mindestens $c/\log n$ für eine geeignete Konstante c. Wir iterieren `FastCut` $\lceil \frac{100}{c} \log n \rceil$ mal. Die Wahrscheinlichkeit, jedesmal *keinen* minimalen Schnitt auszugeben, beträgt dann höchstens $(1 - \frac{c}{\log n})^{\frac{100}{c} \log n} \leq e^{-100} \approx 10^{-43}$. Wir erhalten also einen randomisierten Algorithmus mit Laufzeit $O(n^2 \log^2 n)$ und Erfolgswahrscheinlichkeit $\geq 1 - 10^{-43}$. Der bisher schnellste bekannte deterministische Algorithmus hat eine Laufzeit von $O(ne + n^2 \log n) = O(n^3)$, wobei e die Kantenzahl des Eingabegraphen ist.

2.5 Randomisierte Selektion

Gegeben ist ein Array $S[1, \ldots, n]$. Wir nehmen der Einfachheit halber an, dass alle Elemente verschieden sind und n gerade ist.
Da alle Elemente verschieden sind, können wir jedem der Elemente eindeutig einen Rang zuordnen, und zwar erhält das kleinste Element den Rang 1, das zweitkleinste den Rang 2, und so fort. $S(i)$ bezeichne das Element vom Rang i in S.
Das Selektionsproblem besteht in der Aufgabe, zu gegebenem S und k das Element $S(k)$ zu finden. Wir beschränken uns in diesem Unterkapitel auf den Fall $k = n/2$, d.h. der von uns zu entwerfende Algorithmus soll den Median bestimmen. Den Median kann man offensichtlich deterministisch in Zeit $O(n \log n)$ bestimmen, indem man das gesamte Array mit Heapsort sortiert und dann das Element an Position $n/2$ ausliest.
In der Vorlesung „Datenstrukturen“ bzw. „DAP2“ stellt man einen randomisierten Algorithmus vor, der im Erwartungswert höchstens $4n$ Vergleiche macht, im worst-case aber eben auch quadratisch viele.
Hier wollen wir einen randomisierten Algorithmus vorstellen, der immer mit $2n + o(n)$ vielen Vergleichen auskommt, manchmal allerdings die Antwort „Weiß nicht“ zurückliefert. (Dies allerdings mit einer akzeptablen Wahrscheinlichkeit.)
Zum besseren Verständnis des folgenden Algorithmus beginnen wir mit einem Beispiel. Angenommen, es ist $n = 1000000$ und es ist das Element vom Rang 500000 gesucht. Weiter sei angenommen, dass jemand uns das Element a vom Rang 499000 und das Element b vom Rang 505000 verrät.
Wie könnten wir das ausnutzen? Wir würden das Array S einmal durchlaufen und alle Elemente herausfiltern, die größer gleich a und kleiner gleich b sind. Das wären 6001 viele. Das zu suchende Element vom Rang 500000 befindet sich logischerweise unter diesen.
Nehmen wir an, wir hätten die 6001 Zahlen in ein Array M gefiltert. Dann würde man das Array M sortieren und das Element $S(500000)$ finden, indem man das Element an Position 1001 in M ausliest. Was hat man gewonnen? Man hat (eventuell) eine geringere Laufzeit, weil man das kleinere Array M sortiert hat an Stelle des großen Arrays S.
Das ist die Grundidee hinter folgendem Algorithmus:
Algorithmus select(k, a, b).
Voraussetzung: $a < b$ sind zwei Elemente aus dem Array S.

begin
Bringe mit der Prozedur „Partition“ von Quicksort das Element a an die richtige Stelle in S.
Nennen wir diese Stelle nun $pos(a)$.
Auf dem Teilarray $S[pos(a) + 1], \ldots, S[n]$ bringe das Element b mit der
Prozedur „Partition“ an die richtige Stelle in S.
Falls $pos(a) > k$: **output**(„Weiß nicht. Typ 1.“)
Falls $pos(b) < k$: **output**(„Weiß nicht. Typ 1.“)
Sei $P := pos(b) - pos(a) + 1$.
(P zählt die Anzahl der Elemente, die größer gleich a und kleiner gleich b sind.)
Falls $P > 4n^{3/4} + 2$: **output**(„Weiß nicht. Typ 2.“)
Ansonsten: Sortiere das Array an den Stellen $pos(a), \ldots, pos(b)$ mit Hilfe von Heapsort.

Gib das Element an Stelle k aus.
end
Zwei Bemerkungen: 1.) Die Ausgabe „Typ 1“ bzw. „Typ 2“ haben wir hier nur künstlich hinzugefügt, um die beiden Stellen unterscheiden zu können.
2.) Dass man die Prozedur abbricht, wenn P zu groß ist, soll verhindern, dass man zuviel Laufzeit in das Sortieren des Teilarrays investiert.
Wie kommt man aber an geeignete Werte a und b heran?
Zuerst wird eine Stichprobe aus S zufällig gezogen. Intuitiv sollte klar sein, dass der Median der gesamten Menge S auch mit hoher Wahrscheinlichkeit in der zufälligen Stichprobe nahe der Mitte liegt. Das gibt uns die Intuition, warum wir als a ein Element wählen sollten, das in der Stichprobe ein wenig unterhalb des Stichprobenmedians liegt und als Element b eines, das ein wenig oberhalb des Stichprobenmedians liegt. Genauer gehen wir wie folgt vor:

Algorithmus `LazySelect`

1. Benutze ein Array $R[1, \ldots, n^{3/4}]$. (Die eventuell notwendigen Gaußklammern um das $n^{3/4}$ schenken wir uns.)
2. Für $i = 1, \ldots, n^{3/4}$:
3. **begin** Wähle gemäß der Gleichverteilung ein Element aus dem Array S und speichere es in $R[i]$.
4. **end**
5. Sortiere die Menge R mit HeapSort aufsteigend.
   HeapSort macht $O(|R| \cdot \log |R|) = O(n^{3/4} \cdot \log n) = o(n)$ viele Vergleiche.
6. Für jedes i ist die Wahrscheinlichkeit, dass $R[i] \leq S(k)$ ist, genau k/n. Damit ist $x = \frac{k \cdot n^{3/4}}{n} = k \cdot n^{-1/4}$ die erwartete Anzahl an Elementen in R, die kleiner oder gleich $S(k)$ sind.
7. Wir erwarten also, dass das Element $S(k)$ (würde man es in das Array R einsortieren) an Position x zu stehen käme.
8. Wähle $l = \lfloor x - \sqrt{n} \rfloor$ und $a := R[l]$.
9. Wähle $h = \lceil x + \sqrt{n} \rceil$ und $b := R[h]$.
10. (Für n groß genug ist $1 \leq l$ und $h \leq n$.)
11. Rufe select(k, a, b) auf.

2.22 Satz. *Der Algorithmus* `LazySelect` *führt $2n+o(n)$ viele Vergleiche durch und gibt entweder den Median des Arrays S oder die Meldung „Weiß nicht.“ aus. Die Meldung „Weiß nicht.“ wird dabei mit Wahrscheinlichkeit höchstens $O(\frac{1}{n^{1/4}})$ ausgegeben.*

Beweis: Angenommen, es tritt ein „Weiß nicht“ vom Typ 1 auf. Das kann zwei Gründe haben:

- $a > S(k) \Leftrightarrow a = R[l] > S(k) \Leftrightarrow$ weniger als l Elemente aus R sind kleiner/gleich $S(k)$.
- $b < S(k) \Leftrightarrow b = R[h] < S(k) \Leftrightarrow$ mindestens h Elemente aus R sind kleiner als $S(k)$.

Wenn wir mit X die Zufallsvariable bezeichnen, die angibt, wieviele Elemente in R kleiner oder gleich $S(k)$ sind, dann tritt also die Ausgabe „Weiß nicht. Typ 1.“ auf, wenn $X < l$ oder $X \geq h$ ist. Es macht also Sinn, X genauer anzuschauen. Wir können $X = \sum_{i=1}^{n^{3/4}} X_i$ schreiben, wenn wir die X_i wie folgt definieren:

$$X_i = \begin{cases} 1, \text{ falls } R[i] \leq S(k) \text{ ist} \\ 0, \text{ sonst} \end{cases} .$$

Es ist: $\mathbf{Prob}(X_i = 1) = \frac{k}{n}$, also $\mathbf{E}[X_i] = \frac{k}{n}$. Dies entspricht einem Bernoulliexperiment, dessen Varianz bei Wahrscheinlichkeit p mit $p(1-p)$ errechnet und durch $1/4$ abgeschätzt werden kann (siehe Definition 3.8 und anschließende Rechnung): $\mathbf{V}[X_i] \leq \frac{1}{4}$.
Daraus ergibt sich für den Erwartungswert von X:

$$\mathbf{E}[X] = x = \frac{k}{n} \cdot n^{3/4} = k \cdot n^{-1/4}$$

und für die Varianz, da die X_i unabhängig voneinander sind:

$$\mathbf{V}[X] = \sum_{i=1}^{n^{3/4}} \mathbf{V}[X_i] \leq \frac{n^{3/4}}{4}.$$

Damit erhalten wir:

$$\begin{aligned} \mathbf{Prob}\left(X < l \text{ oder } X \geq h\right) &\leq \mathbf{Prob}\left(|X - x| \geq \sqrt{n}\right) \\ &= \mathbf{Prob}\left(|X - E[X]| \geq \sqrt{n}\right). \end{aligned}$$

An dieser Stelle wenden wir die Ungleichung von Tschebyscheff an. Diese besagte:

$$\forall t > 0: \ \mathbf{Prob}\left(|X - \mathbf{E}[X]| \geq t \cdot \sqrt{\mathbf{V}[X]}\right) \leq \frac{1}{t^2}.$$

Wir wählen $t = \frac{\sqrt{n}}{\sqrt{\mathbf{V}[X]}}$ und erhalten:

$$\mathbf{Prob}\left(|X - E[X]| \geq \sqrt{n}\right) \leq \frac{\mathbf{V}[X]}{n} = O(n^{-1/4}).$$

Damit haben wir die Wahrscheinlichkeit analysiert, mit der der Algorithmus die Antwort „Weiß nicht. Typ 1.“ ausgibt. Die Analyse der Wahrscheinlichkeit, dass der Algorithmus die Antwort „Weiß nicht. Typ 2.“ gibt, wollen wir nur kurz anreißen:

Wähle

$$k_l = k - 2n^{3/4} \text{ sowie } k_h = k + 2n^{3/4}.$$

Für n groß genug ist $k_l \geq 1$ und $k_h \leq n$.
Falls $a \geq S(k_l)$ und $b \leq S(k_h)$, dann gilt $P \leq 4n^{3/4} + 1$, es kommt also nicht zur Meldung „Weiß nicht. Typ 2.“.

$$P > 4n^{3/4} + 2 \Rightarrow P > 4n^{3/4} + 1 \Rightarrow a < S(k_l) \text{ oder } b > S(k_h).$$

Also gilt

$$\mathbf{Prob}\left(P > 4n^{3/4} + 2\right) \leq \mathbf{Prob}\left(a < S(k_l)\right) + \mathbf{Prob}\left(b > S(k_h)\right)$$

und wir begnügen uns mit der Bemerkung, dass man die Wahrscheinlichkeiten $\mathbf{Prob}\left(a < S(k_l)\right)$ und $\mathbf{Prob}\left(b > S(k_h)\right)$ auf ähnliche Weise wie oben analysieren kann, wobei sich auch hier wieder ergibt, dass sie durch $O(n^{-1/4})$ beschränkt sind.

Es ergibt sich eine gesamte Versagenswahrscheinlichkeit von $O(n^{-1/4})$. □

Kapitel 3

Allgemeine Techniken und Prinzipien

3.1 Eine probabilistische Rekurrenz

Bei der Analyse von deterministischen Algorithmen haben wir oft folgende Situation vorliegen: Ein Problem auf n Objekten wird auf ein Problem auf $n/2$ Objekten reduziert, dieses dann auf ein Problem auf $n/4$ Objekten etc. Das Problem auf genau einem Objekt ist dann oft trivial zu lösen.
Es ist klar, dass wir mit $\log n$ Schritten auskommen, bis wir nur noch ein Objekt vorliegen haben, bzw. mit $O(\log n)$ Schritten, wenn n nicht gerade eine Zweierpotenz ist.
Was passiert aber, wenn wir einen randomisierten Algorithmus vorliegen haben, der nach einem Schritt nicht auf *garantiert* exakt $n/2$ Objekte reduziert, sondern nur im Erwartungswert auf $n/2$ Objekte? Intuitiv würden wir immer noch $O(\log n)$ Schritte erwarten, aber trügt uns die Intuition vielleicht? Wir betrachten diese Fragestellung gleich in einer noch allgemeineren Variante.
Es wird eine probabilistische Rekursionsgleichung, die sich bei der Analyse eines sogenannten *random walks* ergibt, beispielhaft gelöst. Diese Lösung kann uns dann bei der Analyse anderer randomisierter Algorithmen hilfreich sein.
Wir betrachten ein Array mit den Positionen $1, 2, \ldots$, in dem sich an einer Position n zu Beginn ein „Partikel“ oder „Teilchen“ befindet. Das Partikel wird an jeder Arrayposition $i > 1$ um eine Zufallsvariable X_i nach links transportiert, und bei $i = 1$ gestoppt. Wir setzen daher voraus, dass $1 \leq X_i \leq i-1$ ist und nehmen an, dass wir von der Zufallsvariable nur wissen, dass $\mathbf{E}[X_i] \geq g(i)$ ist, wobei g eine monoton steigende Funktion ist, die wir kennen. Gesucht ist die Antwort auf die Frage, wie lange es im Erwartungswert dauert, bis das Teilchen an der Position 1 angekommen ist.
Zur Notation: Sei T_n eine Zufallsvariable, die die Anzahl der dafür benötigten Schritte misst. Gesucht ist also $\mathbf{E}[T_n]$. Der folgende Satz liefert eine Antwort auf diese Frage:

3.1 Satz. *Wenn g monoton steigend ist, dann gilt* $\mathbf{E}[T_n] \leq F(n) := \sum_{i=2}^{n} \frac{1}{g(i)}$.

Beweis: Wir zeigen diese Aussage durch vollständige Induktion über n. Der Induktionsanfang ist klar, da $\mathbf{E}[T_1] = 0$ und $F(1) = 0$ ist. Sei jetzt $n \geq 2$. Im Induktionsschritt argumentieren wir wie folgt:

$$\begin{aligned} T_n &= 1 + T_{n-X_n} \Rightarrow \mathbf{E}[T_n] = 1 + \mathbf{E}[T_{n-X_n}] \\ &\leq 1 + \mathbf{E}[F(n - X_n)] \qquad \text{(nach Induktionsvoraussetzung)} \end{aligned}$$

In der letzten Ungleichung benötigen wir, dass jedes $X_n \geq 1$ ist, und außerdem haben wir implizit von einer Eigenschaft Gebrauch gemacht, die im Anhang auf Seite 138 unter der Überschrift „Einschub zu zweistufigen Experimenten“ beschrieben ist.
Wir beobachten nun, dass $F(n) = F(j) + \frac{1}{g(j+1)} + \cdots + \frac{1}{g(n)} = F(n - X_n) + \frac{1}{g(n-X_n+1)} + \cdots + \frac{1}{g(n)}$ ist und können wie folgt umschreiben:

$$\begin{aligned}
&\leq 1 + \mathbf{E}\left[F(n) - \frac{1}{g(n - X_n + 1)} - \cdots - \frac{1}{g(n)}\right] \\
&= 1 + F(n) - \mathbf{E}\left[\frac{1}{g(n - X_n + 1)} + \cdots + \frac{1}{g(n)}\right] \\
&\leq 1 + F(n) - \mathbf{E}\left[X_n \cdot \frac{1}{g(n)}\right] \qquad \text{(wegen der Monotonie von } g\text{)} \\
&= 1 + F(n) - \frac{\mathbf{E}[X_n]}{g(n)} \\
&\leq F(n).
\end{aligned}$$

In der letzten Ungleichung haben wir $\mathbf{E}[X_n] \geq g(n)$ benutzt. □

Im Buch von Motwani und Raghavan wird an dieser Stelle lediglich die schwächere Aussage

$$\mathbf{E}[T_n] \leq \int_1^n \frac{1}{g(x)} dx$$

bewiesen. Diese ergibt sich leicht aus unserem Satz durch die folgende einfache Beobachtung: Wenn $f(x)$ eine monoton fallende Funktion ist, die auf dem Intervall $[a, a+1]$ integrierbar ist, dann gilt

$$\int_a^{a+1} f(x)\, dx \geq \int_a^{a+1} f(a+1) dx = f(a+1), \text{ damit ist}$$

$$\int_1^n \frac{1}{g(x)} dx = \int_1^2 \frac{1}{g(x)} dx + \int_2^3 \frac{1}{g(x)} dx + \cdots + \int_{n-1}^n \frac{1}{g(x)} dx \geq \sum_{i=2}^n \frac{1}{g(i)}.$$

Vollkommen analog gilt natürlich auch: Wenn T_n die Anzahl der Schritte misst, bis das Teilchen in Position 0 (statt Position 1) ist, dann gilt

$$\mathbf{E}[T_n] \leq \sum_{i=1}^n \frac{1}{g(i)}.$$

Anwendungen zur probabilistischen Rekurrenz

`FIND`-Algorithmus

Wenn wir eine Menge von verschiedenen Zahlen gegeben haben, bezeichnen wir mit „Rang“ eines Elements die Position des Elements in der sortierten Reihenfolge, bei n Elementen hat also das kleinste Element den Rang 1 und das größte den Rang n. Das FIND-Problem lässt sich wie folgt beschreiben:

Gegeben: Ein Array S mit n paarweise verschiedenen Zahlen sowie eine Zahl $k \in \{1, \ldots, n\}$.
Gesucht: Das Element mit Rang k.

Las-Vegas-Algorithmus: `FIND`(S,k)

- Wähle ein Element x aus dem Array (gemäß der Gleichverteilung).
- Berechne $S_< := \{y \mid y < x,\ y \text{ im Array}\}$ und $S_> := \{y \mid y > x,\ y \text{ im Array}\}$.
- Fall 1: $|S_<| \geq k$: `FIND`($S_<$,k).
- Fall 2: $|S_<| = k-1$: `output` x.
- Fall 3: $|S_<| \leq k-2$: `FIND`($S_>$,$k - |S_<| - 1$).

3.2 Satz. *Die durchschnittliche Rekursionstiefe für den FIND-Algorithmus ist durch* $4 \cdot H_n$ *beschränkt, wenn er auf einem Array der Größe* n *aufgerufen wird. Dabei ist* H_n *die* n*-te Harmonische Zahl,* $H_n \approx \ln n$.

Beweis: Mit Hilfe des *random walks*: Dabei entspricht die Position des Partikels der Größe der Restmenge. `STOP` spätestens bei Position 1. Damit entspricht die Rekursionstiefe der Anzahl Schritte, bis man an Position 1 angelangt ist. Dank der probabilistischen Rekurrenz reicht es, die Zufallsvariable X_n zu analysieren, die die „verlorenen Elemente“ misst, wenn der Algorithmus auf ein Array aus n Elementen angewendet wird. Sei k_1 der Rang des gewählten Elements x.

- Fall 1: $k_1 > k$. Dann ist $|S_<| \geq k$ und damit fallen $n - k_1 + 1$ viele Elemente weg.
- Fall 2: $k_1 = k$. Also fallen alle $n-1$ anderen Elemente weg.
- Fall 3: $k_1 < k$. Folglich fallen k_1 Elemente weg.

Für den Erwartungswert gilt:

$$\begin{aligned}
\mathbf{E}[X_n] &= \frac{1}{n} \cdot \left(\sum_{k_1=k+1}^{n} (n - k_1 + 1) + n - 1 + \sum_{k_1=1}^{k-1} k_1 \right) \\
&= \frac{1}{n} \cdot \left(\sum_{k_1=1}^{n-k} k_1 + n - 1 + \sum_{k_1=1}^{k-1} k_1 \right) \\
&= \frac{1}{n} \cdot \Big(\frac{(n-k) \cdot (n-k+1)}{2} + n - 1 + \frac{(k-1) \cdot k}{2} \Big) \\
&\geq \frac{1}{n} \cdot \Big(\frac{n^2}{4} - \frac{1}{4} + n - 1 \Big) \geq \frac{n}{4},
\end{aligned}$$

wobei sich das vorletzte Ungleichheitszeichen ergibt, wenn man nach k ableitet und die Ableitung gleich Null setzt. (Ergebnis: $k = \frac{n+1}{2}$.) Die letzte Ungleichheit gilt für $n \geq 2$.
Folglich kann man in der probabilistischen Rekurrenz die Funktion $g(i) = \frac{i}{4}$ verwenden. Daraus ergibt sich

$$\sum_{i=2}^{n} \frac{1}{g(i)} = \sum_{i=2}^{n} \frac{4}{i} = 4 \sum_{i=2}^{n} \frac{1}{i} = 4H_n - 4 \approx 4 \ln n.$$

□

Graph-Färbbarkeit

3.3 Definition. Die *Färbung eines ungerichteten Graphen* $G = (V, E)$ mit k Farben ist eine Abbildung $f : V \to \{1, \ldots, k\}$. Eine *zulässige Färbung eines Graphen* hat zusätzlich die Eigenschaft, dass $f(u) \neq f(v)$ für alle $\{u, v\} \in E$ gilt.

Also bilden alle Knoten, die eine bestimmte Farbe haben, eine unabhängige Menge.

Das Färbbarkeitsproblem lässt sich nun wie folgt beschreiben:

Gegeben: Ein ungerichteter Graph $G = (V, E)$.
Gesucht: Die minimale Farbenanzahl, mit der der Graph G zulässig gefärbt werden kann.

Es ist bekannt, dass das Färbbarkeitsproblem ein NP-hartes Problem ist. Wir zeigen nun, wie man einen einfachen Algorithmus für die Färbung mit Hilfe der probabilistischen Rekurrenz analysieren kann.

3.4 Satz. *Gegeben sei ein ungerichteter Graph $G = (V, E)$. Angenommen, wir haben einen Algorithmus A, der auf jedem Teilgraphen von G, der genau i Knoten enthält, eine unabhängige Menge berechnet, die im Erwartungswert mindestens $g(i)$ Knoten enthält. Dann kann der Graph mit höchstens $\sum_{i=1}^{n} \frac{1}{g(i)}$ vielen Farben zulässig gefärbt werden.*

Beweis: Zum Beweis geben wir einen Algorithmus an:

$k = 0$;
while $G \neq \emptyset$ **do**
begin
  $k := k + 1$;
  Berechne unabhängige Menge I in G. Färbe die Knoten in I mit Farbe k;
  Entferne die Knotenmenge I aus dem Graphen G, also „$G := G - I$“;
end;
Die Anzahl an benutzten Farben ist k.

Die Position des Partikels beim *random walk* entspricht hier der Anzahl der Restknoten im Graphen. Die Anzahl an Schritten bis zur „Position 0“ entspricht der Anzahl der benutzten Farben.
Der Algorithmus berechnet eine Färbung mit k Farben, wobei der Erwartungswert für k durch $\sum_{i=1}^{n} 1/g(i)$ beschränkt ist. Da dies der Erwartungswert für k ist, gibt es insbesondere eine Färbung mit maximal dieser Farbenzahl. □

3.2 Momente, Abweichungen und Okkupanzprobleme

Wir interessieren uns nun nicht nur für Erwartungswerte, sondern auch für Aussagen der folgenden Form: „Mit *hoher Wahrscheinlichkeit* ist die Laufzeit durch T beschränkt.“ Hier stellt sich natürlich die Frage, was wir als hohe Wahrscheinlichkeit akzeptieren. Typischerweise wird hier $1 - o(1)$ gewählt. Andere Möglichkeiten bestehen darin, den $o(1)$-Term zu konkretisieren, also zum Beispiel $1 - 2^{-n}$ oder $1 - n^{-k}$ als hohe Wahrscheinlichkeiten zu bezeichnen.

3.2.1 Einige Definitionen und Begriffe

3.5 Definition. Das k-te zentrale Moment einer Zufallsvariable X (k gerade):

$$\mu^{(k)}(X) := \mathbf{E}\left[(X - \mathbf{E}[X])^k\right].$$

3.6 Definition. Für $k = 2$ ist dies die Varianz, d.h. $\mathbf{V}[X] = \mu^{(2)}(X)$.

3.7 Satz. *a) Wenn X eine Zufallsvariable ist, so gilt*

$$\mathbf{V}[X] = \mathbf{E}\left[X^2\right] - \mathbf{E}[X]^2.$$

b) Wenn X eine Zufallsvariable mit Werten aus $\{0,1\}$ ist, dann gilt

$$\mathbf{V}[X] = \mathbf{E}[X] \cdot (1 - \mathbf{E}[X]).$$

Beweis:

$$\begin{aligned} \text{Es ist } \mathbf{V}[X] &= \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \\ &= \mathbf{E}\left[X^2 - 2X \cdot \mathbf{E}[X] + \mathbf{E}[X]^2\right] \\ &= \mathbf{E}\left[X^2\right] - 2\mathbf{E}[X]^2 + \mathbf{E}[X]^2 \\ &= \mathbf{E}\left[X^2\right] - \mathbf{E}[X]^2 . \end{aligned}$$

Wenn X nur Werte aus $\{0,1\}$ annimmt, dann gilt $X = X^2$ und somit $\mathbf{E}\left[X^2\right] = \mathbf{E}[X]$ und die Aussage b) ergibt sich durch Ausklammern. □

3.8 Definition. Ein Bernoulliexperiment ist ein Zufallsexperiment mit der Zufallsvariable

$$X = \begin{cases} 1 & \text{mit Wahrscheinlichkeit } p \\ 0 & \text{sonst} \end{cases} .$$

3.9 Behauptung. Die Varianz eines Bernoulliexperiments X ist

$$\mathbf{V}[X] = p(1-p) \leq 1/4.$$

Dies ergibt sich direkt aus dem letzten Satz, Teil b), da X eine $\{0,1\}$-Zufallsvariable ist.

3.10 Behauptung (Linearität der Varianz bei paarweiser Unabhängigkeit)**.** Wenn $X_1, \ldots, X_n$ paarweise unabhängige Zufallsvariablen sind und $X := \sum_{i=1}^{n} X_i$ ist, dann ist

$$\mathbf{V}[X] = \sum_{i=1}^{n} \mathbf{V}[X_i] .$$

Beweis: Sei zur Schreibvereinfachung $\mu_i := \mathbf{E}[X_i]$, $\mu := \mathbf{E}[X]$, also $\mu = \sum_{i=1}^{n} \mu_i$. Ebenso sei $\mathrm{ab}_i := X_i - \mu_i$. (Den Namen ab benutzen wir als Abkürzung für „Abweichung“.) Natürlich ist $\mathbf{E}[\mathrm{ab}_i] = \mathbf{E}[X_i] - \mu_i = 0$ und $\mathbf{V}[X_i] = \mathbf{E}\left[\mathrm{ab}_i^2\right]$.

$$\begin{aligned} \mathbf{V}[X] &= \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \\ &= \mathbf{E}\left[\left(\sum_{i=1}^{n} X_i - \sum_{i=1}^{n} \mu_i\right)^2\right] \\ &= \mathbf{E}\left[\left(\sum_{i=1}^{n} \mathrm{ab}_i\right)^2\right] \\ &= \mathbf{E}\left[(\mathrm{ab}_1 + \cdots + \mathrm{ab}_n)^2\right] \\ &= \mathbf{E}\left[\mathrm{ab}_1^2 + \cdots + \mathrm{ab}_n^2 + 2 \cdot \mathrm{ab}_1 \cdot \mathrm{ab}_2 + \cdots + 2 \cdot \mathrm{ab}_{n-1} \cdot \mathrm{ab}_n\right] . \end{aligned}$$

Da X_i und X_j für $i \neq j$ unabhängig voneinander sind, sind auch ab_i und ab_j unabhängig voneinander, daher gilt:

$$\begin{aligned} \mathbf{E}[\mathrm{ab}_i \cdot \mathrm{ab}_j] &= \mathbf{E}[\mathrm{ab}_i] \cdot \mathbf{E}[\mathrm{ab}_j] \\ &= 0. \end{aligned}$$

Also ist

$$\begin{aligned}\mathbf{V}[X] &= \mathbf{E}\left[\mathrm{ab}_1^2 + \cdots + \mathrm{ab}_n^2\right] \\ &= \mathbf{E}\left[\mathrm{ab}_1^2\right] + \cdots + \mathbf{E}\left[\mathrm{ab}_n^2\right] \\ &= \mathbf{V}[X_1] + \cdots + \mathbf{V}[X_n].\end{aligned}$$

□

3.3 Two-Point-Sampling

Hier direkt eine „Anwendung“, bei der ausgenutzt wird, dass die Linearität der Varianz bei paarweiser Unabhängigkeit gilt.

3.11 Definition. Zufallsvariablen $Y_1, \ldots, Y_N$ heißen genau dann paarweise unabhängig, wenn für alle $i \neq j$ die Zufallsvariablen Y_i und Y_j unabhängig sind.

3.12 Lemma. *Sei p eine Primzahl und Z_p die abelsche Gruppe modulo p. Seien Zufallsvariablen $Y_0, \ldots, Y_{p-1}$ wie folgt definiert: wähle $a, b \in Z_p$ gemäß der Gleichverteilung und setze für alle $i \in \{0, \ldots, p-1\}$ die Zufallsvariable $Y_i = a \cdot i + b \bmod p$. Dann gilt: $Y_0, \ldots, Y_{p-1}$ sind paarweise unabhängig und jedes Y_j nimmt Werte aus $\{0, \ldots, p-1\}$ gemäß der Gleichverteilung an.*

Bevor wir das Lemma beweisen, zunächst einmal ein *Beispiel*:
Die folgenden Matrizen enthalten alle vorkommenden Möglichkeiten für den Fall $p = 5$ und $i = 0, 1, 2$, jede Spalte entspricht einem Wert von b, jede Zeile einem Wert von a. Die erste Matrix steht für die Belegung der Zufallsvariable Y_0, die zweite für die Belegung der Zufallsvariable Y_1, etc.

0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
0	1	2	3	4	1	2	3	4	0	2	3	4	0	1
0	1	2	3	4	2	3	4	0	1	4	0	1	2	3
0	1	2	3	4	3	4	0	1	2	1	2	3	4	0
0	1	2	3	4	4	0	1	2	3	3	4	0	1	2

Die Zufallsvariable Y_i ist durch die i-te Matrix definiert: Zeile und Spalte werden ausgewürfelt und Y_i nimmt den Wert an der entsprechenden Position an. Paarweise Unabhängigkeit von zum Beispiel Y_0 und Y_1 bedeutet hier, dass wenn wir alle Werte für a und b durchlaufen, an den entsprechenden Stellen in den Matrizen alle Wertepaare genau einmal vorfinden. Zum Beispiel für $(a, b) = (0, 0)$ sehen wir in Matrix Y_0 den Wert 0 und in Matrix Y_1 den Wert 0, für $(a, b) = (2, 2)$ sehen wir die beiden Werte $(2, 4)$, für $(a, b) = (4, 2)$ das Paar $(2, 1)$, etc.

Zum Beweis des Lemmas

Beweis: Zu zeigen ist, dass für alle $c_1, c_2 \in Z_p$ gilt:

$$\mathbf{Prob}\left(Y_i = c_1 \text{ und } Y_j = c_2\right) = \mathbf{Prob}\left(Y_i = c_1\right) \cdot \mathbf{Prob}\left(Y_j = c_2\right).$$

Wir zeigen zunächst, dass die rechte Seite den Wert $1/p^2$ ergibt:

$$\begin{aligned}\mathbf{Prob}\left(Y_i = c_1\right) &= \mathbf{Prob}\left(a \cdot i + b \bmod p = c_1\right) \\ &= \mathbf{Prob}\left(b \equiv c_1 - a \cdot i \bmod p\right) \\ &= 1/p.\end{aligned}$$

Die Wahrscheinlichkeit ist $1/p$, weil nach der Wahl von a genau eine Wahl von b dafür sorgt, dass die obige Kongruenz erfüllt ist. Das Produkt der Wahrscheinlichkeiten für Y_i und Y_j ergibt

$1/p^2$.

Nun zur linken Seite: Wie groß ist die Wahrscheinlichkeit, dass $Y_i = c_1$ und $Y_j = c_2$ ist?
Das Polynom $f(x) := ax + b$ ist ein Polynom vom Grad 1 über dem Körper Z_p. Ein solches Polynom liegt nach dem Interpolationstheorem eindeutig fest, wenn wir seinen Wert an zwei Stellen kennen.
Wenn $Y_i = c_1$ und $Y_j = c_2$ ist, dann ist $f(i) = c_1$ und $f(j) = c_2$, somit liegt das Koeffizientenpaar a, b eindeutig fest. Jedes Koeffizientenpaar wird aber mit Wahrscheinlichkeit $1/p^2$ ausgewürfelt.
□

Anwendung

Paarweise unabhängige Variablen können benutzt werden, um RP-Algorithmen zu amplifizieren.
Gegeben ein RP-Algorithmus A, der wie folgt funktioniert:

x	Eingabe
r	Zufallswert $\in Z_p$

würfle $r \in Z_p$ aus und gib $A(x, r)$ aus, wobei $A(x, r)$ zur Rate-Verifikations-Maschine deterministisch arbeitet. (Hier fühlt man sich an den Begriff der Rate-Verifikations-Maschine aus der Vorlesung GTI erinnert.)
Da A ein RP-Algorithmus ist, gilt:

$$x \in L \Rightarrow Prob_r(A(x, r) = 1) \geq \tfrac{1}{2}$$
$$x \notin L \Rightarrow Prob_r(A(x, r) = 0) = 1$$

Um r auszuwürfeln, braucht man $O(\log p)$ Bits. Bei N-facher Iteration kann man den Fehler des RP-Algorithmus auf 2^{-N} drücken, dabei verwendet man $O(N \log p)$ Zufallsbits. Wenn man also mit $O(\log p)$ Zufallsbits auskommen möchte, kann man nur einen Fehler von $O(1)$ erzielen.
„Gute“ Zufallszahlen zu erzeugen kann aufwändig sein. Das folgende Vorgehen spart Zufallsbits: Wir würfeln zwei Zufallszahlen $a \in Z_p$ und $b \in Z_p$ aus und berechnen für $i = 0, \ldots, N-1$ die Zahlen $r_i := a \cdot i + b \bmod p$.
Unser neuer Algorithmus, nennen wir ihn A^*, ruft der Reihe nach $A(x, r_i)$ auf und gibt am Ende eine Eins aus, wenn für $Y := \sum_{i=0}^{N-1} A(x, r_i)$ gilt, dass $Y > 0$ ist.
Jedes $A(x, r_i)$ ist eine Zufallsvariable mit Trägermenge $\{0, 1\}$. Damit ist die Varianz der Zufallsvariable maximal $1/4$. Ihr Erwartungswert ist mindestens $1/2$, wenn $x \in L$ ist und ist gleich 0 sonst.

$$x \in L \Rightarrow \mathbf{E}[Y] = \sum_{i=0}^{N-1} \mathbf{E}[A(x, r_i)] \geq N/2.$$
$$x \notin L \Rightarrow \mathbf{E}[Y] = 0.$$

Da nach dem letzten Lemma die $r_0, \ldots, r_{N-1}$ und somit die $A(x, r_i)$ paarweise unabhängig sind, gilt:

$$\mathbf{V}[Y] = \sum_{i=0}^{N-1} \mathbf{V}[A(x, r_i)] \leq \frac{N}{4}.$$

Also gilt

$$\begin{aligned} \mathbf{Prob}(A^* \text{ rechnet auf } x \in L \text{ falsch}) &= \mathbf{Prob}(Y = 0) \\ &\leq \mathbf{Prob}(|Y - \mathbf{E}[Y]| \geq \mathbf{E}[Y]) \\ &\leq \frac{\mathbf{V}[Y]}{\mathbf{E}[Y]^2} \leq \frac{N/4}{N^2/4} = \frac{1}{N}. \end{aligned}$$

Dabei haben wir wieder die Tschebyscheffsche Ungleichung verwendet, diesmal mit $t := \frac{\mathbf{E}[Y]}{\sqrt{\mathbf{V}[Y]}}$.

Fazit: Bei dem neuen Vorgehen erreicht man mit $O(\log p)$ Zufallsbits bei $N \leq p$ Iterationen eine Fehlerwahrscheinlichkeit, die höchstens $1/N$ ist. Man kann also eine Fehlerwahrscheinlichkeit von $1/p$ erreichen.
Allerdings sollte man N auch nicht zu groß wählen, also zum Beispiel nicht $N = p$ wählen, da man bei p Iterationen auch gleich alle Möglichkeiten $r \in \{0, \ldots, p-1\}$ durchprobieren könnte.

3.3.1 Das Coupon Collector Problem

Gegeben sind n Boxen $1, \ldots, n$ und Bälle, die gemäß der Gleichverteilung auf die Boxen verteilt werden. Wie groß ist die *erwartete* Anzahl der Ballwürfe, bis jede Box mindestens einen Ball enthält? Hier besteht ein Zusammenhang zum Lottoproblem (siehe Seite 10). Beim Lottoproblem lautete die Frage, wie viele Zufallsversuche (= Ziehung einer Zahl) wir durchführen müssen, bis 6 verschiedene Zahlen gezogen wurden. Wenn wir 6 von n Kugeln ziehen wollen, dann lautet die Antwort:

$$\frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{n-5}.$$

Das Coupon Collector Problem lässt sich also auch so formulieren: Wieviele Zufallsversuche X erwarten wir, bis n verschiedene Kugeln gezogen wurden? Dieser Erwartungswert $\mathbf{E}[X]$ errechnet sich als

$$\sum_{i=1}^{n} \frac{n}{i} = n \cdot \sum_{i=1}^{n} \frac{1}{i} = n \cdot H_n.$$

Damit ist $n \ln n + 1 \leq \mathbf{E}[X] \leq n \ln n + n$. (Siehe Satz A.4.)

3.13 Satz. *Für jede Konstante $\beta > 1$ gilt: Nach mehr als $\beta n \ln n$ Ballwürfen ist die Wahrscheinlichkeit, dass keine Box leer ist,* $1 - o(1)$.

Beweis: Betrachte die Box i. Die Wahrscheinlichkeit, dass Box i leer ist, beträgt nach mindestens $\beta n \ln n$ Ballwürfen höchstens

$$\left(1 - \frac{1}{n}\right)^{\beta n \ln n} \leq e^{-\beta \ln n} = n^{-\beta}.$$

Die Wahrscheinlichkeit, dass mindestens eine Box leer ist, ist also kleiner als $n^{-\beta+1} = o(1)$. Die Wahrscheinlichkeit, dass alle Boxen mindestens einen Ball enthalten, ist also mindestens $1 - o(1)$. □

Andererseits kann man auch fragen, wie voll typischerweise die Box mit den meisten Bällen ist. Bevor wir diese Frage beantworten, geben wir noch einige Abschätzungen ohne Beweis an, die wir hierfür benötigen:

1. $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$
2. $\binom{n}{k} \leq \frac{n^k}{k!}$
3. $\binom{n}{k} \leq \left(\frac{e \cdot n}{k}\right)^k$

Sei nun n die Anzahl der Boxen und m die Anzahl der Ballwürfe, $E_{\geq k}$ das Ereignis, dass Box 1 mindestens k Bälle enthält. Dann gilt (siehe Anhang):

$$\mathbf{Prob}\left(E_{\geq k}\right) \leq \binom{m}{k} \cdot \left(\frac{1}{n}\right)^k.$$

Die Wahrscheinlichkeit, dass mindestens eine Box mindestens k Bälle enthält, ist höchstens

$$n\binom{m}{k}\left(\frac{1}{n}\right)^k \leq n\left(\frac{e \cdot m}{k}\right)^k \left(\frac{1}{n}\right)^k \quad \text{(wegen Abschätzung 3).}$$

Setze $m = n$, dann ist dies höchstens

$$n\left(\frac{e\cdot n}{k}\right)^k\left(\frac{1}{n}\right)^k \leq n\left(\frac{e}{k}\right)^k$$

Wähle $k := \lceil \frac{e\ln n}{\ln\ln n}\rceil$. Einsetzen von k ergibt

$$n\left(\frac{e}{k}\right)^k \leq n\left(\frac{\ln\ln n}{\ln n}\right)^k = ne^{-k\ln\ln n + k\ln\ln\ln n} \leq ne^{-e\ln n + \lceil \frac{e\ln n}{\ln\ln n}\rceil \ln\ln\ln n}$$

Beachte, dass für $n \geq n_0$ gilt, dass

$$\frac{\lceil \frac{e\ln n}{\ln\ln n}\rceil \ln\ln\ln n}{\ln n} \leq 0.7.$$

Also ist

$$ne^{-e\ln n + \lceil \frac{e\ln n}{\ln\ln n}\rceil \ln\ln\ln n} \leq ne^{-e\ln n + 0{,}7\ln n} \leq nn^{-2} = \frac{1}{n} = o(1)$$

Damit ist die Wahrscheinlichkeit, dass eine Box mindestens $\lceil \frac{e\ln n}{\ln\ln n}\rceil$ Elemente enthält, $o(1)$. Wir halten fest:

3.14 Satz. *Sei $max_{n,n}$ die größte Füllung einer Box nach n Ballwürfen in n Boxen. Dann ist*

$$\mathbf{Prob}\left(max_{n,n} \geq \frac{e\cdot\ln n}{\ln\ln n}\right) = o(1).$$

In Worten: Mit hoher Wahrscheinlichkeit haben alle Boxen eine durch $e\cdot\ln n/\ln\ln n$ beschränkte Füllhöhe.

3.3.2 Das Geburtstagsparadoxon

Wieviele Personen benötigt man, bis die Wahrscheinlichkeit, dass 2 der Personen am gleichen Tag Geburtstag haben, mindestens 1/2 beträgt? Wir nehmen eine Gleichverteilung der Geburtsdaten über 365 Tage an.

Sei E_i das Ereignis, dass Person i an einem anderen Tag Geburtstag hat als die Personen $1, 2, \ldots, i-1$. Dann gilt:

$$\begin{aligned}
& \mathbf{Prob}\,(m \text{ Personen haben an verschiedenen Tagen Geburtstag}) \\
= \;& \mathbf{Prob}\,(E_2 \wedge \cdots \wedge E_m) \\
= \;& \mathbf{Prob}\,(E_2)\cdot\mathbf{Prob}\,(E_3 \mid E_2)\cdot\mathbf{Prob}\,(E_4 \mid E_3\wedge E_2)\cdot\cdots\mathbf{Prob}\,(E_m \mid E_2\wedge\cdots\wedge E_{m-1}) \\
= \;& \prod_{i=2}^{m}\left(1-\frac{i-1}{n}\right) \leq \prod_{i=2}^{m} e^{-\frac{i-1}{n}} = e^{-\frac{1+2+\cdots+(m-1)}{n}} = e^{-\frac{m(m-1)}{2n}}.
\end{aligned}$$

Wähle $m \geq \sqrt{2n}+1$, dann ist $\frac{m(m-1)}{2n} \geq 1$. Damit ergibt sich $e^{-\frac{-m(m-1)}{2n}} \leq e^{-1} < \frac{1}{2}$. Wenn $n = 365$ ist, ist $\sqrt{2n}+1 \approx 28$.

3.4 Das Prinzip der verzögerten Entscheidung

Das Prinzip der verzögerten Entscheidung ist ein Analyseverfahren, das in der folgenden Situation angewendet werden kann: Gegeben ist ein Algorithmus A, dessen Verhalten auf zufälligen Inputs untersucht werden soll, der aber für festen Input deterministisch ist. Bisher haben wir in diesem Fall eine Eingabe $x = (x_1, \ldots, x_n)$ vorweg komplett ausgewürfelt und dann die deterministische Laufzeit auf dieser Eingabe analysiert. Beim Prinzip der verzögerten Entscheidung wird nun jeder Teil des Inputs dann einzeln ausgewürfelt, wenn er gebraucht wird. Wenn man dafür sorgt, dass in beiden Fällen die gleiche Wahrscheinlichkeitsverteilung vorliegt, kann man diesen Algorithmus A_{lazy} anstelle des Algorithmus A analysieren.

Beispiel

Zur Motivierung des Prinzips der verzögerten Entscheidung betrachten wir das folgende Kartenspiel: Gegeben sind 52 handelsübliche Spielkarten, je 13 der Farben $\diamondsuit, \heartsuit, \spadesuit, \clubsuit$. Diese werden gemischt und gleichmäßig auf vier Stapel verteilt. Die Stapel bekommen als symbolische Namen ebenfalls die Bezeichner $\diamondsuit, \heartsuit, \spadesuit, \clubsuit$. Dies bedeutet aber keineswegs, dass (z.B.) im Stapel $\clubsuit$ nur Karten der Farbe $\clubsuit$ sind. Den Spielablauf beschreiben wir im Pseudocode:

```
z := ♢; counter := 1
while Stapel mit Farbe z not empty
      ziehe Karte K vom Stapel mit Farbe z
      z := Farbe von K
      counter++
```

Das Spiel ist gewonnen, wenn `counter = 52`, und sonst verloren.

Man startet also mit dem Karostapel, nimmt von dort eine Karte. Die Farbe dieser Karte sagt einem, von welchem Stapel man die nächste Karte ziehen soll. Und so fort. Das Spiel endet, wenn man versucht, auf einen leeren Stapel zuzugreifen. Man verliert, wenn beim Spielende nicht alle Karten vom Tisch sind.
Übrigens endet das Spiel immer mit einem versuchten Zugriff auf den Karostapel, denn wenn ein Stapel leer wird, der nicht der Karostapel ist, dann hat man von der Farbe dieses Stapels dreizehn Karten gesehen und es kann danach nicht passieren, dass man noch einmal auf den Stapel zugreifen muss. Der Karostapel spielt deswegen eine besondere Rolle, weil der erste Zugriff auf ihn „von außen“ vorgegeben ist.
Die interessante Frage ist nun, wie groß die Gewinnwahrscheinlichkeit ist. Diese Analyse ist ohne die Methode der verzögerten Entscheidung nicht sehr elegant durchzuführen, da die Stapelverteilung, also alle 52 Karten vorab zufällig erzeugt werden müsste. Wir erzeugen jetzt aber die Farbe einer Karte erst dann, wenn sie gezogen wird. Damit entspricht ein Spiel einer Folge von 52 Kartenfarben bzw. einer Verteilung von 52 Karten in einer bestimmten Reihenfolge.
Wir gewinnen genau dann, wenn wir die dreizehnte Karokarte ziehen und der Tisch anschließend leer ist. Dies ist genau dann der Fall, wenn die dreizehnte Karokarte die letzte der 52 Karten ist. Also beträgt die Gewinnwahrscheinlichkeit genau $\frac{1}{4}$.

3.4.1 Stabile Heiraten

Wir betrachten als Anwendung das *Problem der stabilen Heiraten*: Gegeben sind n Männer, n Frauen, und Permutationen $\pi_1, \ldots, \pi_n$ sowie $\sigma_1, \ldots, \sigma_n$. Die Interpretation ist die folgende: Die Permutationen π_i geben die Reihenfolge an, in der der Mann i die Frauen mag, analog geben die Permutationen σ_i die Reihenfolge an, in der die Frau i die Männer mag.
Gesucht ist nun ein Matching bzw. eine Menge von Heiraten $(1, a_1), (2, a_2), \ldots, (n, a_n)$, so dass jeder Mann mit genau einer Frau verheiratet ist und die Heiraten stabil sind, d.h. es dürfen keine zwei Paare $(i_1, j_1), (i_2, j_2)$ existieren mit der Eigenschaft, dass Mann i_1 die Frau j_2 seiner Frau j_1 bevorzugt und umgekehrt j_2 den Mann i_1 ihrem Mann i_2 bevorzugt. Eine Menge von Heiraten, die stabil ist, wird auch einfacher als stabile Heirat bezeichnet.
Die berühmteste Anwendung für dieses Problem stammt aus den USA, wo Medizinstudierende für ein Praktikum auf Krankenhäuser verteilt werden. (Siehe z.B. `http://www.ams.org/featurecolumn/archive/marriage.html`)

Beispiel Wir bezeichnen die Männer zur besseren Unterscheidung mit den Großbuchstaben A, B, C, D, die Frauen mit kleinen Buchstaben a, b, c, d (dies liegt lediglich in der erwarteten Körpergröße begründet) und betrachten folgende Vorlieben:

$$\begin{array}{llll} A: abcd & B: bacd & C: adcb & D: dcab \\ a: ABCD & b: DCBA & c: ABCD & d: CDAB \end{array}$$

Das Matching $(A,a),(B,b),(C,c),(D,d)$ ist nicht stabil, da C d gegenüber c bevorzugt und d C gegenüber D. Eine Lösung dieser Instanz ist das stabile Matching $(A,a),(B,b),(C,d),(D,c)$, dies wird gleich bewiesen werden.
Vorsicht: Dieses Beispiel suggeriert, mit einem lokalen *greedy*-Ansatz könne das Problem gelöst werden. Dies ist im Allgemeinen NICHT möglich!

3.15 Satz. *Für jede Eingabe $\pi_1,\ldots,\pi_n,\sigma_1,\ldots,\sigma_n$ gibt es eine stabile Heirat. Eine solche kann mit dem folgenden Algorithmus* `Proposal` *(Heiratsantrag) berechnet werden.*

Algorithmus `Proposal`
Der Algorithmus folgt im wesentlichen dem *greedy*-Ansatz, jedoch verheiratet er Paare zunächst nur vorläufig. Treten Instabilitäten auf, so wird eine vorläufige Entscheidung rückgängig gemacht.

`while exists` unverheirateter Mann M `do`
    M wählt unter allen Frauen, die noch keinen Heiratsantrag von ihm erhalten haben, die ihm sympathischste Frau F aus. $(*)$
    M macht F einen Heiratsantrag.
    Falls F unverheiratet ist, akzeptiert sie den Heiratsantrag.
    Falls F verheiratet ist und M ihrem aktuellen Mann gegenüber bevorzugt, akzeptiert sie den Heiratsantrag und lässt sich von ihrem aktuellen Mann scheiden.

Beispiel (Fortsetzung) Wir lassen den Algorithmus zur Illustration einmal auf der Eingabe des letzten Beispiels laufen:

1. A macht a einen Heiratsantrag $\rightarrow (A,a)$
2. C macht a einen Heiratsantrag $\rightarrow$ a lehnt ab
3. C macht d einen Heiratsantrag $\rightarrow (C,d)$
4. D macht d einen Heiratsantrag $\rightarrow$ d lehnt ab
5. D macht c einen Heiratsantrag $\rightarrow (D,c)$
6. B macht b einen Heiratsantrag $\rightarrow (B,b)$

Analyse des Algorithmus und Beweis des Satzes

- Die Liste in $(*)$ ist nie leer, denn alle Frauen bleiben verheiratet, falls sie einmal verheiratet sind. Sie wechseln höchstens ihren Partner. Falls alle Frauen den Heiratsantrag von M abgelehnt haben, waren sie zu dem Zeitpunkt verheiratet. Wenn n Frauen verheiratet sind, müssen deshalb auch n Männer verheiratet sein, sonst ergäbe sich ein Widerspruch zur Monogamieforderung.
- Der Algorithmus ist deterministisch.
- Die worst-case-Laufzeit ist durch $O(n^2)$ beschränkt. Bei jedem Durchlauf der while-Schleife findet ein Antrag statt. Dabei wird in der Liste des jeweiligen Mannes eine Frau von der Liste gestrichen, nämlich diejenige, der der Antrag gemacht wurde. Da auf jeder Liste n Frauen stehen, kann die while-Schleife also nur maximal n^2 mal durchlaufen werden.

  Dass dieser worst-case auch (zumindest asymptotisch) eintreten kann, zeigt der Fall, dass alle $\sigma_i = \pi_i = (1,2\ldots,n)$ sind: Wenn die Männer in der Reihenfolge 1 bis n ihre Heiratsanträge machen dürfen, dann macht Mann Nummer i den Frauen 1 bis i Heiratsanträge, wobei die ersten $i-1$ davon aber abgelehnt werden. Das macht zusammen $1+2+\cdots+n = \Theta(n^2)$ Heiratsanträge.

- Die Korrektheit zeigen wir per Widerspruchsbeweis. Wir nehmen an, es gäbe ein Heiratspaar $(X, x), (Y, y)$, das instabil ist. Dann bevorzugt X Frau y gegenüber x. Also machte X Frau y den Antrag eher als x. Wir argumentieren nun, dass y einen Mann haben muss, den sie lieber mag als X. Wir unterscheiden zwei Fälle:

  Fall 1 Der Antrag von X an y wurde von y abgelehnt. Dann war y zum Zeitpunkt des Heiratsantrags mit einem ihr sympathischeren Mann als X verheiratet und somit auch später, also ist ihr Y lieber als X. Widerspruch.

  Fall 2 Der Antrag von X an y wurde von y angenommen. Da y später neu geheiratet hat, muß sie Y lieber haben als X. Widerspruch.

Eine average-case-Analyse Unser Ziel ist es nun, die durchschnittliche Anzahl an Heiratsanträgen zu analysieren. Dazu analysieren wir das Verhalten auf einem zufälligen Input $\pi_1, \ldots, \pi_n, \sigma_1, \ldots, \sigma_n$, der gemäß der Gleichverteilung ausgewürfelt wird. Das Prinzip der verzögerten Entscheidung bedeutet hier, dass die Reihenfolgen erst dann ausgewürfelt werden, wenn sie relevant sind, d.h. immer dann, wenn ein Mann einen Heiratsantrag macht, würfelt er die ihm liebste Frau unter allen noch nicht gefragten aus. Dann gilt:

$$\textbf{Prob}\,(\text{modifizierter Algo macht } T \text{ Anträge}) = \textbf{Prob}\,(\texttt{Proposal} \text{ macht } T \text{ Anträge})$$

Um die Analyse zu vereinfachen, betrachten wir anstelle dieses modifizierten Algorithmus den Algorithmus `Amnesie`, bei dem ein Mann die Frau, der er einen Antrag macht, aus allen Frauen auswürfelt. Dieser Algorithmus macht offensichtlich höchstens mehr Schritte. Wir erhalten:

$$\begin{aligned}&\textbf{Prob}\,(\texttt{Amnesie} \text{ macht } \geq T \text{ Anträge})\\ \geq\ &\textbf{Prob}\,(\text{modifizierter Algo macht } T \text{ Anträge})\\ =\ &\textbf{Prob}\,(\texttt{Proposal} \text{ macht } T \text{ Anträge})\end{aligned}$$

Mit der Interpretation „Frauen entsprechen den Boxen“ und „Heiratsantrag ist ein Ballwurf in eine Box“ wird klar, dass folgendes gilt:
Behauptung: X sei die Zufallsvariable, die die Anzahl der Ballwürfe misst, bis jede von n Boxen mindestens einen Ball enthält. Y sei die Zufallsvariable, die die Anzahl der Heiratsanträge angibt, bis alle verheiratet sind. Offenbar ist $X = Y$, da beiden Zufallsvariablen die gleiche Grundmenge und die gleiche Wahrscheinlichkeitsverteilung zugrunde liegt. Das Problem lässt sich also als Coupon Collector Problem auffassen, für das wir gezeigt haben, dass $\mathbf{E}[X] = n \ln n + O(n)$ gilt und wo für alle $\beta > 1$ gilt, dass $\textbf{Prob}\,(X \geq \beta n \ln n) \leq o(1)$ ist.
Wir haben gezeigt:

3.16 Satz. *Der Algorithmus Proposal löst das Problem der stabilen Heiraten bei zufälligem Input in erwarteter Laufzeit $O(n \log n)$.*

Zurück zum Coupon-Collector-Problem Wir zeigen zunächst eine einfach zu beweisende, dann eine genauere, aber schwieriger zu beweisende Aussage.

3.17 Behauptung. X sei die Zufallsvariable, die die Anzahl der Versuche misst, bis jede Box mindestens einen Ball enthält. Dann gilt für alle $c > 0$:

$$\textbf{Prob}\,(|X - \mathbf{E}[X]| \geq c \cdot n) \leq \frac{\pi^2}{6 \cdot c^2} \leq \frac{1.65}{c^2}.$$

Beweis: Sei X_i die Anzahl Versuche, die man braucht, um bei $i-1$ schon belegten Boxen einen Ball in eine freie Box zu werfen. Da die Wahrscheinlichkeit, eine freie Box zu erwischen, $p = \frac{n-i+1}{n}$ ist, ist $\mathbf{E}[X_i] = 1/p = \frac{n}{n-i+1}$.
Für die Zufallsvariable X gilt $X = \sum_{i=1}^{n} X_i$ und wegen der Unabhängigkeit der X_i haben wir für die Varianz

$$\mathbf{V}[X] = \mathbf{V}[X_1] + \cdots + \mathbf{V}[X_n],$$

also interessiert nun die Varianz $\mathbf{V}[X_i]$. Wir halten zunächst fest, dass X_i gleich der Anzahl der Versuche ist, bis Erfolg eintritt und dieser pro Versuch mit Wahrscheinlichkeit p eintritt. Folglich gilt $\mathbf{Prob}(X_i = j) = (1-p)^{j-1}p$. Daraus ergibt sich:

$$\begin{aligned}
\mathbf{V}[X_i] &= \mathbf{E}\left[X_i^2\right] - \mathbf{E}[X_i]^2 \\
&= (\sum_{j=1}^{\infty} \mathbf{Prob}(X_i = j) \cdot j^2) - (1/p^2). \\
&= (\sum_{j=1}^{\infty} (1-p)^{j-1} \cdot p \cdot j^2) - (1/p^2). \\
&= \cdots (\text{kleinere Rechnung ausgelassen}) \cdots \\
&= \frac{1-p}{p^2} \\
&\leq \frac{1}{p^2}.
\end{aligned}$$

Wenn wir nun $p = (n-i+1)/n$ einsetzen, so erhalten wir:

$$\mathbf{V}[X_i] \leq \frac{n^2}{(n-i+1)^2}.$$

Damit erhalten wir insgesamt:

$$\begin{aligned}
\mathbf{V}[X] &= \sum_{i=1}^{n} \mathbf{V}[X_i] \\
&\leq \sum_{i=1}^{n} \frac{n^2}{(n-i+1)^2} \\
&= n^2 \cdot \sum_{i=1}^{n} \frac{1}{i^2} \\
&\leq n^2 \cdot \frac{\pi^2}{6}.
\end{aligned}$$

wobei das letzte Kleinergleich von der Tatsache $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$ herrührt.
Wegen der Tschebyscheff-Ungleichung — $\mathbf{Prob}\left(|X - \mathbf{E}[X]| \geq t\sqrt{\mathbf{V}[X]}\right) \leq \frac{1}{t^2}$ — erhalten wir (bei Wahl von $t = c \cdot n/\sqrt{\mathbf{V}[X]}$):

$$\mathbf{Prob}(|X - \mathbf{E}[X]| \geq c \cdot n) \leq \frac{\mathbf{V}[X]}{c^2 \cdot n^2} \leq \frac{\pi^2}{c^2 \cdot 6} \leq \frac{1.65}{c^2}.$$

□

Nun zur präziseren Aussage:

3.18 Behauptung. X sei die Zufallsvariable, die die Anzahl der Versuche misst, bis jede Box mindestens einen Ball enthält. Dann gilt:

$$\forall c \in \mathbb{R}^+ : \lim_{n\to\infty} \mathbf{Prob}(n \ln n - cn \leq X \leq n \ln n + cn) = e^{-e^{-c}} - e^{-e^{c}}.$$

Bei z.B. $c = 4$ ist $\mathbf{Prob}(\ldots) \approx 0.98$.
Beweis: Wir benötigen das Prinzip der Inklusion/Exklusion. Für zwei Mengen A und B besagt dieses:

$$\mathbf{Prob}(A \cup B) = \mathbf{Prob}(A) + \mathbf{Prob}(B) - \mathbf{Prob}(A \cap B),$$

für drei Mengen

$$\begin{aligned}\mathbf{Prob}\,(A \cup B \cup C) = \quad & \mathbf{Prob}\,(A) + \mathbf{Prob}\,(B) + \mathbf{Prob}\,(C) \\ - \quad & \mathbf{Prob}\,(A \cap B) - \mathbf{Prob}\,(A \cap C) - \mathbf{Prob}\,(B \cap C) \\ + \quad & \mathbf{Prob}\,(A \cap B \cap C)\,,\end{aligned}$$

bzw. allgemeiner, wenn wir $P_k^n := \sum_{S \subseteq \{1,\ldots,n\}, |S|=k} \mathbf{Prob}\left(\bigcap_{i \in S} E_i\right)$ abkürzen:

$$\mathbf{Prob}\,(E_1 \cup E_2 \cdots \cup E_n) = P_1^n - P_2^n + P_3^n - \cdots = \sum_{k=1}^{n} (-1)^{k+1} P_k^n.$$

Desweiteren benötigen wir die Ungleichung von Boole-Bonferroni, die wir hier nicht beweisen wollen. Sie besagt im Prinzip, dass wir eine untere Schranke für die Wahrscheinlichkeit $P_1^n - P_2^n + P_3^n - \cdots$ bekommen, wenn wir nach einem Minuszeichen abbrechen und eine obere Schranke erhalten, wenn wir nach einem Plus abbrechen. Man ist ja nicht besonders überrascht, dass so etwas gilt:

$$\begin{aligned} k \text{ gerade}: \quad & \mathbf{Prob}\,(\textstyle\bigcup_{i=1}^n E_i) \geq \sum_{j=1}^k (-1)^{j+1} P_j^n \\ k \text{ ungerade}: \quad & \mathbf{Prob}\,(\textstyle\bigcup_{i=1}^n E_i) \leq \sum_{j=1}^k (-1)^{j+1} P_j^n. \end{aligned}$$

Sei E_i^m das Ereignis, dass Box i bei m Ballwürfen leer bleibt, also ist $\mathbf{Prob}\,(X > m) = \mathbf{Prob}\,(\bigcup_{i=1}^n E_i^m) = \sum_{k=1}^n (-1)^{k+1} P_k^n$.
Am liebsten würden wir nun die Wahrscheinlichkeit

$$\mathbf{Prob}\,(E_1^m \cup \cdots \cup E_n^m)$$

genau ausrechnen. Da das aber schwierig ist, geben wir uns mit einer asymptotischen Aussage zufrieden. Sei

$$S_k^n := P_1^n - P_2^n + P_3^n - \cdots \pm P_k^n,$$

mit $S_0^n = 0$.
S_k^n ist der Wert, den man erhält, wenn man die Formel für $\mathbf{Prob}\,(E_1^m \cup \cdots \cup E_n^m)$ nach k Termen abbricht. S_k^n ist also eine Schätzung der gesuchten Wahrscheinlichkeit. Für uns ist von Interesse, wie genau die Schätzung ist.
Die Ungleichung von Boole-Bonferroni besagt, dass für alle n und alle $k \geq 0$ mit $2k+1 \leq n$ folgendes gilt:

$$S_{2k}^n \leq \mathbf{Prob}\,(E_1^m \cup \cdots \cup E_n^m) \leq S_{2k+1}^n.$$

Wenn es uns also gelingt, nachzuweisen, dass für alle k der Wert $S_k := \lim_{n\to\infty} S_k^n$ existiert, dann wissen wir, dass gilt:

$$\forall k : S_{2k} \leq \lim_{n\to\infty} \mathbf{Prob}\,(E_1^m \cup \cdots \cup E_n^m) \leq S_{2k+1}.$$

Wenn wir dann zeigen können, dass $\lim_{k\to\infty} S_k$ existiert (und zum Beispiel den Wert T annimmt), dann gilt auch

$$\lim_{k\to\infty} S_{2k} \leq \lim_{n\to\infty} \mathbf{Prob}\,(E_1^m \cup \cdots \cup E_n^m) \leq \lim_{k\to\infty} S_{2k+1},$$

also

$$T \leq \lim_{n\to\infty} \mathbf{Prob}\,(E_1^m \cup \cdots \cup E_n^m) \leq T$$

und somit

$$T = \lim_{n\to\infty} \mathbf{Prob}\,(E_1^m \cup \cdots \cup E_n^m)\,.$$

Nun kümmern wir uns um die Berechnung der Limiten: Zunächst beobachten wir, dass $P_j^n = \binom{n}{j}(1 - \frac{j}{n})^m$ ist. Hier ergibt sich der erste Term, weil wir j der insgesamt n Boxen auswählen müssen; von denen wir in m Versuchen keine auswählen dürfen, dies besagt der zweite Term.

Man kann (unter Verwendung der bekannten Abschätzungen für die Binomialfunktion) zeigen, dass $\lim_{n\to\infty} P_j^n = \frac{e^{-cj}}{j!} =: P_j$ ist, wenn $m = n\ln n + c\cdot n \pm O(1)$ ist (c eine reelle Konstante). Daraus erhält man:

$$\begin{aligned} S_k := \lim_{n\to\infty} S_k^n &= \lim_{n\to\infty} P_1^n - P_2^n + \cdots \pm P_k^n \\ &= P_1 - P_2 + \cdots \pm P_k \\ &= \frac{e^{-c1}}{1!} - \frac{e^{-c2}}{2!} + \frac{e^{-c3}}{3!} - \cdots \pm \frac{e^{-ck}}{k!}. \end{aligned}$$

Wir erinnern daran, dass die Reihenentwicklung der Exponentialfunktion

$$\exp(x) := e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^i}{i!} + \cdots$$

ist. Nun erkennt man direkt, dass die Formel für S_k nichts anderes ist als die ersten k Terme der Reihenentwicklung von $1 - \exp(-e^{-c})$. Wir erhalten $\lim_{k\to\infty} S_k = 1 - e^{-e^{-c}}$.
Da $\mathbf{Prob}(X > m) = \mathbf{Prob}(X > \lceil n\ln n + cn\rceil) = \mathbf{Prob}(X > n\ln n + cn)$ ist (X ist auf jeden Fall eine ganze Zahl), haben wir nun gezeigt, dass

$$\lim_{n\to\infty} \mathbf{Prob}(X > n\ln n + cn) = 1 - e^{-e^{-c}}$$

ist. Ebenso (mit $-c$ statt c und nach untem gerundetem m) folgt, dass $\lim_{n\to\infty} \mathbf{Prob}(X \geq n\ln n - cn) = 1 - e^{-e^{c}}$ ist.
Die Wahrscheinlichkeit, dass $n\ln n - cn \leq X \leq n\ln n + c$ ist, ergibt sich also aus der Differenz $\mathbf{Prob}(X \geq n\ln n - cn) - \mathbf{Prob}(X > n\ln n + cn)$ und somit

$$\lim_{n\to\infty} \mathbf{Prob}(n\ln n - cn \leq X \leq n\ln n + cn) = e^{-e^{-c}} - e^{-e^{c}}.$$

□

3.5 Chernoffschranke(n)

$X_1, X_2, \ldots, X_n$ seien n unabhängige Zufallsvariablen mit $X_i \in \{0,1\}$ und $\mathbf{Prob}(X_i = 1) = p_i$. Dann gilt für $X = X_1 + X_2 + \cdots + X_n$ und $\mu := \mathbf{E}[X] = \sum_{i=1}^n p_i$:

a0) Für alle $\delta \geq 0$ und alle $m \geq \mu$ gilt	:	$\mathbf{Prob}(X \geq (1+\delta)\cdot m) \leq \left[\frac{e^\delta}{(1+\delta)^{1+\delta}}\right]^m$
a1) Für alle $\delta \geq 0$ gilt	:	$\mathbf{Prob}(X \geq (1+\delta)\cdot \mu) \leq \left[\frac{e^\delta}{(1+\delta)^{1+\delta}}\right]^\mu$
a2) Für alle $0 \leq \delta < 1$ gilt	:	$\mathbf{Prob}(X \leq (1-\delta)\cdot \mu) \leq \left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^\mu$
a3) Für alle $0 \leq \delta$ gilt	:	$\mathbf{Prob}(X \leq (1-\delta)\cdot \mu) \leq e^{-\mu\frac{\delta^2}{3}}$
b) Für alle $0 \leq \delta < 1$ und alle $m \geq \mu$ gilt	:	$\mathbf{Prob}(X \geq (1+\delta)\cdot m) \leq e^{-m\frac{\delta^2}{3}}$
c) Für alle $R \geq 5\mu$ gilt	:	$\mathbf{Prob}(X \geq R) \leq (1/2)^R$
d) Für alle $0 \leq \delta < 1$ gilt	:	$\mathbf{Prob}(\lvert X - \mu\rvert \geq \delta\cdot\mu) \leq 2\cdot e^{-\mu\delta^2/3}$.

(**Anmerkung:** Man kann sich relativ leicht davon überzeugen, dass die Funktion $\frac{e^x}{(1+x)^{1+x}}$ für alle $x > -1$ größer als 0 und kleiner gleich 1 ist. Daher sind die auf der rechten Seite

vorkommenden Zahlen $\left[\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right]$ und $\left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]$ kleiner gleich 1.)

Nun zum Beweis der Chernoffschranke(n):
Beweis: Die Aussage a1) ergibt sich aus Aussage a0) durch Einsetzen von $m := \mu$.
Aussage c) erhält man aus a0) wie folgt: Setze $\delta := 4$ und $m := R/5$. Da $R \geq 5\mu$ ist, ist $m \geq \mu$ und daher kann a0) angewendet werden. Einsetzen ergibt als rechte Seite der Schranke:

$$\left(\frac{e^4}{5^5}\right)^{R/5} = \left(\frac{e^{4/5}}{5}\right)^{R} \leq 0.45^R \leq (1/2)^R.$$

Aussage d) ergibt sich wie folgt. Der Fall $|X - \mu| \geq \delta \cdot \mu$ tritt genau dann auf, wenn entweder $X \geq (1+\delta) \cdot \mu$ oder $X \leq (1-\delta) \cdot \mu$ ist. Wegen der bekannten Eigenschaft $\mathbf{Prob}\,(A \cup B) \leq \mathbf{Prob}\,(A) + \mathbf{Prob}\,(B)$ folgt die Aussage d) nun aus a3) und b).
Für den Beweis zu den anderen Aussagen machen wir zunächst eine Vorüberlegung. Wir wählen ein $t \in \mathbf{R}$ und berechnen $\mathbf{E}\left[e^{t \cdot X}\right]$ für die Zufallsvariable X:

$$\begin{aligned}
\mathbf{E}\left[e^{t\cdot X}\right] &= \mathbf{E}\left[e^{t\cdot(X_1+X_2+\cdots+X_n)}\right] = \mathbf{E}\left[\prod_{i=1}^{n} e^{t\cdot X_i}\right] = \prod_{i=1}^{n} \mathbf{E}\left[e^{t\cdot X_i}\right] \\
&= \prod_{i=1}^{n}(1\cdot(1-p_i)+p_i e^t) = \prod_{i=1}^{n}(1+p_i(e^t-1)) \\
&\leq e^{p_1(e^t-1)}\cdots e^{p_n(e^t-1)} \\
&= \left(e^{p_1+p_2+\cdots+p_n}\right)^{e^t-1} \\
&= e^{\mu(e^t-1)}.
\end{aligned}$$

In der Ungleichung haben wir die Eigenschaft $1 + x \leq e^x$ auf $x = p_i(e^t - 1)$ angewendet.

Zu a0) Für $\delta = 0$ ist die Aussage eine triviale. Wir können also im folgenden $\delta > 0$ annehmen. Wenn $t > 0$ eine (reelle) Zahl ist, so können wir wie folgt rechnen:

$$\begin{aligned}
\mathbf{Prob}\,(X \geq (1+\delta)\cdot m) &\overset{t>0}{=} \mathbf{Prob}\left(e^{tX} \geq e^{t(1+\delta)\cdot m}\right) \\
&\overset{\text{Markov}}{\leq} \frac{\mathbf{E}\left[e^{tX}\right]}{e^{t(1+\delta)\cdot m}} \leq \frac{e^{\mu(e^t-1)}}{e^{(1+\delta)\cdot mt}} \\
&\leq \frac{e^{m(e^t-1)}}{e^{(1+\delta)\cdot mt}} \rightsquigarrow \text{wähle } t = \ln(1+\delta) > 0 \\
&\leq \frac{e^{m\cdot(1+\delta-1)}}{e^{(1+\delta)\cdot m\ln(1+\delta)}} = \frac{e^{\delta\cdot m}}{(1+\delta)^{(1+\delta)\cdot m}}.
\end{aligned}$$

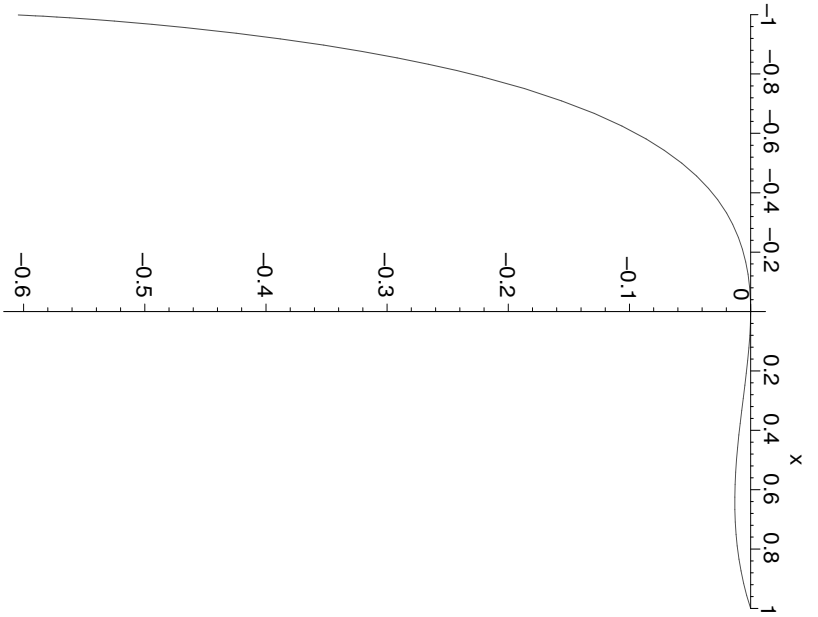
zu a2) Für $\delta = 0$ ist die Aussage wieder trivial, sei also $\delta > 0$ im folgenden. Für jede reelle Zahl $t < 0$ gilt:

$$\begin{aligned}
\mathbf{Prob}\,(X \leq (1-\delta)\cdot \mu) &\overset{t<0}{=} \mathbf{Prob}\left(e^{tX} \geq e^{t(1-\delta)\cdot\mu}\right) \\
&\leq \cdots \\
&\leq \frac{e^{(e^t-1)\mu}}{e^{(1-\delta)\cdot\mu t}} \rightsquigarrow \text{wähle } t = \ln(1-\delta) < 0 \\
&\leq \left[\frac{e^{(1-\delta-1)}}{e^{(1-\delta)\ln(1-\delta)}}\right]^{\mu} = \left[\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right]^{\mu}
\end{aligned}$$

zu b) Wir schreiben $\frac{e^x}{(1+x)^{1+x}} = e^{x-(1+x)\cdot\ln(1+x)}$ und weisen nach, dass für die Konstante $c = 2\ln 2 - 1 \approx 0.386$ und $x \in (-1, 1]$ gilt, dass

$$x - (1+x)\cdot\ln(1+x) \leq -c\cdot x^2$$

gilt. Man kann dies mit einer klassischen Kurvendiskussion nachweisen. Dies tun wir im Anhang, Satz A.6. Hier wollen wir uns die Gültigkeit der Ungleichung durch einen Funktionenplot zumindest plausibel machen. Der folgende Funktionsgraph zeigt für $f(x) := x - (1+x)\ln(1+x)$ und $g(x) := -x^2\,(2\ln(2) - 1)$ den Verlauf von $f(x) - g(x)$.



Es gilt also für alle $x \in (-1, 1]$, dass $\frac{e^x}{(1+x)^{1+x}} \leq e^{-x^2\cdot c}$ ist. Da $c \approx 0.386 > 1/3$ ist, ergibt sich also Aussage b) aus a1).
Übrigens kann man eine bessere Konstante bekommen, wenn man sich auf den Fall beschränkt, dass x negativ ist.
Für Aussage a3) haben wir zumindest den Fall $0 \leq \delta < 1$ aus a2) hergeleitet. Für $\delta \geq 1$ machen wir die folgenden Überlegungen: Wenn $\delta = 1$ ist, dann erhalten wir im Beweis zu a2) die Aussage $\mathbf{Prob}\,(X \leq (1-\delta)\cdot\mu) \leq e^{(e^t-1)\mu}$. Da man t beliebig negativ wählen darf, ergibt sich daraus für $\delta = 1$ die Aussage $\mathbf{Prob}\,(X \leq (1-\delta)\cdot\mu) \leq e^{-\mu} \leq e^{-\mu/3}$.
Wenn $\delta > 1$ ist, dann kann das Ereignis $X \leq (1-\delta)\cdot\mu$ nicht eintreten, da X eine nicht-negative Zufallsvariable ist. Also ist $\mathbf{Prob}\,(X \leq (1-\delta)\cdot\mu) = 0$ und die Abschätzung in a3) ist eine triviale Aussage. □

3.19 Beispiel. (Anwendung der Chernoffschranke:)
Gegeben seien n Münzwürfe, also n Zufallsvariablen $X_1, \ldots, X_n \in \{0, 1\}$ mit $\mathbf{Prob}\,(X_i = 1) = 1/2$. Sei $X := \sum_{i=1}^{n} X_i$ die Anzahl der Einsen. Dann gilt für alle $n \geq 20$:

$$\mathbf{Prob}\left(|X - \frac{n}{2}| > \frac{1}{2}\sqrt{6n\ln n}\right) \leq \frac{2}{n}.$$

Beweis: $\mu = \mathbf{E}\,[X] = \frac{n}{2}$. Wähle $\delta := \sqrt{\frac{6\ln n}{n}}$. Für $n \geq 20$ ist dies kleiner als 1. Es gilt nach d):

$$\begin{aligned}\mathbf{Prob}\left(|X-\mu| \geq \frac{1}{2}\sqrt{6n\ln n}\right) &= \mathbf{Prob}\,(|X-\mu| \geq \delta\cdot\mu)\\ &\leq 2\cdot e^{-\frac{6\ln n}{3n}\cdot\frac{n}{2}} = 2\cdot e^{-\ln n} = \frac{2}{n}.\end{aligned}$$

□

Hoeffdingsche Ungleichung

Auf ähnliche Art und Weise kann man die Hoeffdingsche Ungleichung beweisen, wir geben hier allerdings nur die Aussage an:

3.20 Satz (Hoeffding)**.** *Sei Z eine Zufallsvariable, die Werte zwischen a und b annimmt und den Erwartungswert μ hat. Seien $Z_1, \ldots, Z_m$ unabhängige Ausprägungen von Z, dann gilt für ihren Durchschnitt $M_m := (Z_1 + \cdots + Z_m)/m$:*

$$\forall c \geq 0 : \mathbf{Prob}\left(|M_m - \mu| \geq c\right) \leq 2e^{-2m\left(\frac{c}{b-a}\right)^2}.$$

Anwendung der Ungleichungen

Im folgenden werden die einzelnen Ungleichungen auf das obige Beispiel angewendet:

$$\begin{aligned}
\text{Markov} \quad & \mathbf{Prob}\left(X \geq \tfrac{3}{4}n\right) \leq \tfrac{E[X]}{(3/4)\cdot n} = \tfrac{\frac{n}{2}}{\frac{3}{4}n} = \tfrac{2}{3} \\
\text{Tschebyscheff} \quad & \mathbf{Prob}\left(X \geq \tfrac{3}{4}n\right) \leq \mathbf{Prob}\left(|X - (n/2)| \geq n/4\right) \leq \tfrac{\mathbf{V}[X]}{(n/4)^2} \leq \tfrac{4}{n} \\
\text{Chernoff} \quad & \mathbf{Prob}\left(X \geq \tfrac{3}{4}n\right) \leq e^{-(\frac{1}{2})^2 \cdot \frac{1}{3} \cdot \frac{1}{2}n} = e^{-\frac{1}{24}n}
\end{aligned}$$

(Bei Tschebyscheff haben wir $\mathbf{V}[X] \leq n/4$ benutzt.)
Man sieht, dass die (asymptotische) Genauigkeit mit den wachsenden Einschränkungen zunimmt. Für die Markov-Ungleichung gelten keine Einschränkungen, für Tschebyscheff müssen die Zufallsvariablen paarweise voneinander unabhängig sein und damit man die Chernoffschranke anwenden darf, müssen alle Zufallsvariablen unabhängig voneinander sein.
Wir führen nun ein repräsentatives Beispiel für Rechnungen mit der Chernoffschranke vor. Mit $||v||_\infty = ||(v_1, \ldots, v_n)||_\infty := \max_i\{|v_i|\}$ bezeichnen wir im folgenden die „Unendlich-Norm“:

3.21 Beispiel. *Set Balancing*: Gegeben sei eine $n \times n$-Matrix A mit Einträgen aus $\{0, 1\}$. Gesucht ist ein Vektor $b \in \{-1, 1\}^n$ mit $||A \cdot b||_\infty$ minimal.

Set Balancing ist als NP-hartes Problem bekannt.
Wir bemerken vorweg, dass für jedes b die Ungleichung $||A \cdot b||_\infty \leq n$ gilt.

3.22 Satz. *Sei $n \geq 2$. Für jede $n \times n$-Matrix A mit Einträgen aus $\{0, 1\}$ gibt es einen Vektor $b \in \{-1, 1\}^n$ mit*

$$||A \cdot b||_\infty \leq \sqrt{12n \ln n}.$$

Beweis: Für $n = 2$ gilt die Aussage trivialerweise, da dann $\sqrt{12n \ln n} > n$ ist. Sei also $n \geq 3$. Wir wählen ein $b \in \{-1, 1\}^n$ gemäß der Gleichverteilung und zeigen, dass folgendes gilt:

$$\mathbf{Prob}\left(||A \cdot b||_\infty > \sqrt{12n \ln n}\right) \leq \frac{2}{n} < 1.$$

Damit wissen wir um die Existenz eines b, das die Eigenschaften des Satzes hat, und es ist auch klar, dass wir ein solches b randomisiert schnell bekommen können.
Der i-te Eintrag im Ergebnis $A \cdot b$ ergibt sich durch Multiplikation der i-ten Zeile a_i der Matrix A mit dem Vektor b. Sei E_i das Ereignis, dass

$$|a_i \cdot b| > \sqrt{12n \ln n}$$

ist. Dann ist

$$\begin{aligned}
\mathbf{Prob}\left(||A \cdot b||_\infty > \sqrt{12n \ln n}\right) &= \mathbf{Prob}\left(E_1 \cup E_2 \cup \cdots \cup E_n\right) \\
&\leq \mathbf{Prob}\left(E_1\right) + \mathbf{Prob}\left(E_2\right) + \cdots + \mathbf{Prob}\left(E_n\right).
\end{aligned}$$

Wir analysieren nun also die Wahrscheinlichkeit $\mathbf{Prob}(E_i)$. Im folgenden kürzen wir $a := a_i$ ab. Die Anzahl der Einsen in a heiße k, o.B.d.A. nehmen wir an, dass die k Einsen in a an den ersten k Stellen stehen.
Für das Ergebnis der Multiplikation $a \cdot b$ sind offensichtlich nur die ersten k Stellen von b relevant, da an den anderen Stellen a Nullen hat.
Sei X die Zufallsvariable, die die Anzahl Einsen zählt, die b an den Stellen $1, \ldots, k$ hat.
$a \cdot b$ ist eine Zufallsvariable, die den Wert $X - (k - X) = 2X - k$ hat, denn an den Positionen, an denen a eine 1 hat, hat b genau X Einsen und genau $k - X$ „Minuseinsen“. Damit gilt $a \cdot b = 2X - k$ und $|a \cdot b| = |2X - k|$.
Wir wollen in wenigen Augenblicken die Chernoffschranke mit $\delta := \frac{2 \cdot \sqrt{3n \ln n}}{k}$ bemühen. Damit wir dies tun können, müssen wir garantieren, dass $\delta < 1$ ist. Wir machen daher eine Fallunterscheidung:
Wenn $k \leq 2 \cdot \sqrt{3n \ln n}$ ist, dann gilt trivialerweise $|a \cdot b| \leq k \leq \sqrt{12n \ln n}$ und $\mathbf{Prob}(E_i) = 0 \leq 2/n^2$.
Sei also nun $k > 2 \cdot \sqrt{3n \ln n}$, dann ist $\delta < 1$. Wir halten zunächst fest, dass $\mu = \mathbf{E}[X] = k/2$ ist. Es ergibt sich also:

$$\begin{aligned}
\mathbf{Prob}\left(|a \cdot b| > \sqrt{12n \ln n}\right) &= \mathbf{Prob}\left(|2X - k| > \sqrt{12n \ln n}\right) \\
&= \mathbf{Prob}\left(|X - (k/2)| > \sqrt{3n \ln n}\right) \\
&= \mathbf{Prob}\left(|X - \mu| > \sqrt{3n \ln n}\right) \\
&= \mathbf{Prob}\left(|X - \mu| > \delta \cdot \frac{k}{2}\right) \\
&\leq \mathbf{Prob}\left(|X - \mu| \geq \delta \cdot \frac{k}{2}\right) \\
&= \mathbf{Prob}\left(|X - \mu| \geq \delta \cdot \mu\right) \\
&\leq 2 \cdot e^{-\frac{\delta^2}{3}\mu} \\
&= 2 \cdot e^{-\frac{12n \ln n}{3k^2} \cdot \frac{k}{2}} \\
&= 2 \cdot e^{-2\frac{n \ln n}{k}} \\
&\leq 2 \cdot e^{-2 \ln n} = 2/n^2.
\end{aligned}$$

Also ist $\mathbf{Prob}(E_i) = \mathbf{Prob}\left(|a \cdot b| > \sqrt{12n \ln n}\right) \leq 2/n^2$ und $\mathbf{Prob}(E_1) + \cdots + \mathbf{Prob}(E_n) \leq 2/n$. □

Kapitel 4

Fortgeschrittene Techniken

4.1 Routing bei Parallelrechnern

Dem Problem liegt folgendes Modell zugrunde: Gegeben sind N Prozessoren und Verbindungen (= Kommunikationskanäle) zwischen den Prozessoren. Wir modellieren dies als einen gerichteten Graphen $G = (V, E)$. Jede Kante kann pro Zeiteinheit eine Informationseinheit (Paket) übertragen. Das System ist also getaktet und in jedem Schritt kann ein Prozessor Nachrichten an alle Nachbarn schicken. Pro Kante gibt es eine Queue, die Pakete aufnehmen kann.

4.1.1 Permutation Routing Problem

Zu Beginn hat jeder Prozessor genau ein Paket i. Zu jedem Paket i ist sein Zielort (ebenfalls ein Prozessor) angegeben, diesen bezeichnen wir mit $d(i)$. Die Bezeichnung „d“ beruht auf dem englischen Namen „destination“.
Frage: Wie viele Schritte braucht man, um die Pakete an ihre Zielorte zu schicken ($\rightsquigarrow$ routen)?
Problem: Mehrere angekommene Pakete müssen über die gleiche Kante verschickt werden.

4.1 Definition. Ein *OPRA* (= Oblivious Permutation Routing Algorithm) ist ein Algorithmus, der festlegt, wohin ein Paket weiterverschickt wird. „Oblivious“ kann mit „stereotyp“ übersetzt werden und bedeutet, dass die Route von Paket i nur von $d(i)$ und nicht von den Zielorten $d(j), j \neq i$, der anderen Pakete abhängt.

Für OPRAs gilt folgende Aussage, die wir hier nicht beweisen:

4.2 Satz. *Gegeben sei ein Netzwerk mit N Knoten, in dem jeder Knoten Grad d hat. Dann gilt: Für jeden deterministischen OPRA auf diesem Netzwerk existiert eine Permutation, so dass der OPRA $\Omega\left(\sqrt{\frac{N}{d}}\right)$ viele Schritte benötigt, um diese Permutation zu routen.*

Der boolesche Hyperwürfel mit 2^n Knoten $v \in \{0,1\}^n$ ist ein Beispiel für eine typische Netzwerktopologie. Eine Kante zwischen v und w existiert genau dann, wenn sich v und w an genau einer Stelle unterscheiden. Hier ist $d = n$ und $N = 2^n$, jeder deterministische OPRA benötigt folglich im worst-case $\Omega\left(\sqrt{\frac{2^n}{n}}\right)$ Schritte, also exponentiell viele. Ein möglicher OPRA für den booleschen Hyperwürfel ist der *bit-fixing* Algorithmus.

Algorithmus `bit-fixing`

- Wenn sich ein Paket i am Knoten $v \in \{0,1\}^n$ befindet und sein Ziel $d(i)$ ist, dann sei j^* die erste Position „von links“, an der sich die Vektoren v und $d(i)$ unterscheiden.
- v^* entsteht aus v, indem das Bit j^* geflippt wird.

- Schicke das Paket über die Kante (v, v^*).

Als Beispiel hier der Weg eines Pakets, das sich am Knoten 0110 befindet und den Zielknoten 1000 hat: $0110 \to 1110 \to 1010 \to 1000$. Bei diesem Verfahren wird jedes Paket offensichtlich über höchstens n Knoten geschickt. Allerdings kann es zu Wartezeiten an den Knoten kommen. Um ein Gefühl dafür zu bekommen, wie Beweise unterer Schranken aussehen, begnügen wir uns mit einem Beispiel einer unteren Schranke für den bit-fixing-Algorithmus, die etwas schlechter ist als die weiter oben angegebene allgemeinere Schranke.

4.3 Behauptung. Es gibt eine Permutation auf dem booleschen Hyperwürfel mit 2^n Knoten, so dass folgendes gilt: Der bit-fixing Algorithmus benötigt mindestens $\Omega\left(\frac{\sqrt{2^n}}{n}\right)$ viele Schritte, um die Permutation zu routen.

Beweis: O.B.d.A. sei n gerade. Wir wählen die Permutation *„transpose“*, die die Eigenschaft hat, dass

$$d(a_1, \ldots, a_n) := (a_{n/2+1}, \ldots, a_n, a_1, \ldots, a_{n/2})$$

ist, der Zielort entsteht also aus dem Startort durch Vertauschen der vorderen und hinteren „Hälfte“ des Vektors. Betrachte die $2^{n/2}$ vielen Positionen $(b_1, \ldots, b_{n/2}, 0, \ldots, 0)$. Diese haben als Zielorte $(0, \ldots, 0, b_1, \ldots, b_{n/2})$ und es ist klar, dass der bit-fixing-Algorithmus die entsprechenden $2^{n/2}$ vielen Pakete alle über den Nullvektor routet. Da nur ein Paket den Nullvektor auch als Ziel hat, müssen $2^{n/2} - 1$ viele Pakete den Nullvektor auch wieder verlassen. Da den Nullvektor nur n Kanten verlassen, gibt es eine Kante, über die $\frac{2^{n/2}-1}{n}$ viele Pakete fließen. □

Wir betrachten nun folgenden randomisierten OPRA („Oblivious“ bedeutet bei *randomisierten* OPRAs, dass die Wahrscheinlichkeitsverteilung über die Routen nur von $d(i)$ abhängt):

- 1. Phase: Wähle zufällige Zwischenstationen $\sigma(i) \in \{1, \ldots, N\}$. Schicke i via bit-fixing nach $\sigma(i)$.
- Wartephase: Falls ein Paket nach weniger als $4n$ Zeiteinheiten bei $\sigma(i)$ angekommen ist, so wartet es, bis $4n$ Zeiteinheiten verstrichen sind.
- 2. Phase: Schicke Pakete mit bit-fixing von ihrer Zwischenstation $\sigma(i)$ nach $d(i)$.

(Beachte, dass σ nicht notwendigerweise eine Permutation ist, da $\sigma(i) = \sigma(j)$ für $i \neq j$ möglich ist.) Als Queueing-Strategie an den Knoten wählen wir FIFO (falls mehrere Pakete zur gleichen Zeit eingetroffen sind, legen wir deren Reihenfolge beliebig fest).

4.1.2 Analyse der erwarteten Laufzeit

Wir analysieren nun die Laufzeit der ersten Phase. Betrachte Paket i und seine Route ρ_i. Die Anzahl der Schritte des Pakets ergibt sich aus der Länge von ρ_i plus der Anzahl der Verzögerungen, die das Paket in den Queues erlebt hat.

Beobachtung: Wenn ρ_1 und ρ_2 zwei Routen sind, die gemäß der bit-fixing Strategie gewählt werden und die gemeinsame Kanten haben, dann gilt: Nachdem sie auseinander gelaufen sind, begegnen sie sich nicht noch einmal.

Beweis: Sei w ein gemeinsamer Knoten beider Routen, den sie in verschiedene Nachfolgerknoten $w_1 \neq w_2$ verlassen. Der Vektor w_1 entsteht aus w durch Flippen des Bits l_1 und w_2 entsteht aus w durch Flippen des Bits l_2, o.B.d.A. liegt l_1 links von l_2.

$$\boxed{\begin{array}{cccccccc} 0 & 1 & 0 & 0 & 1 & 1 & \ldots \\ & & & l_1 & & l_2 & \end{array}} \to \begin{cases} \text{Flippe } l_1 \to & \boxed{0\,1\,0\ \overset{l_1}{1}\ 1\ \overset{l_2}{1}\ \ldots} \\ \text{Flippe } l_2 \to & \boxed{0\,1\,0\ \overset{l_1}{0}\ 1\ \overset{l_2}{0}\ \ldots} \end{cases}$$

Da bit-fixing immer das linkeste Bit flippt, werden in beiden Vektoren die Positionen 1 bis l_1 nicht mehr verändert. Da sich die Vektoren in der Position l_1 unterscheiden, und diese Position im weiteren Verlauf unverändert bleibt, unterscheiden sich auch alle nachfolgenden Knoten in der Position l_1. Daher haben die beiden Routen anschließend keinen Knoten mehr gemeinsam. □

4.4 Lemma. *Die Route von Paket i sei $\rho_i = (e_1, \ldots, e_k)$. S sei die Menge der Pakete ($\neq i$), die auch über mindestens eine der Kanten $e_1, \ldots, e_k$ laufen. Dann gilt: Paket i wird höchstens $|S|$ mal verzögert.*

Beweis: Wir betrachten den Weg eines beliebigen Pakets i. Die Menge $S^* \subseteq S \bigcup \{i\}$ enthalte alle Pakete, die sich zu einem bestimmten Zeitpunkt in einer Queue zu den Kanten $e_1, \ldots, e_k$ befinden. Falls $s \in S^*$ sei $q(s)$ die Nummer der Kante, in deren Queue das Paket s liegt. Wir definieren den „Lag-Wert“ (zu deutsch vielleicht „Verzögerungswert“) eines Pakets s zu einem bestimmten Zeitpunkt t als $lag(s) := t - q(s)$. Das Paket i hat zu Beginn den lag-Wert 0 ($t = 1, q(i) = 1$). Am Ende des Algorithmus stimmt der lag-Wert von i mit der Anzahl der Verzögerungen überein, die das Paket erfährt. Wir zeigen nun, dass am Ende des Algorithmus $lag(i) \leq |S|$ gilt.
Wir benutzen hierzu einen Buchhaltertrick. Wir verbuchen die Kosten für eine Erhöhung des lag-Werts von Paket i auf den Konten von $|S|$ vielen Paketen. Angenommen, der lag-Wert von Paket i steigt von l auf $l + 1$, dann gibt es ein Paket $j \neq i$, das anschließend den lag-Wert l hat. Sei t' der letzte Zeitpunkt, zu dem irgendein Paket aus S den lag-Wert l hat. Es gibt ein Paket v, das ρ_i zum Zeitpunkt t' verlässt. Dieses Paket bekommt die Kosten 1 zugewiesen. Insgesamt bekommt dann jedes der $|S|$ vielen Pakete maximal Kosten 1 zugewiesen. (Jedes Paket kann die Route ρ_i nur einmal verlassen, wie die Beobachtung weiter oben gezeigt hat.) □

Wir definieren nun Zufallsvariablen, die messen, ob zwei Pakete $i \neq j$ Routen haben, die mindestens eine Kante gemeinsam haben.

4.5 Definition. Für $i \neq j$ definiere $H_{ij} := \begin{cases} 1, & \text{falls die Routen } \rho_i \text{ und } \rho_j \\ & \text{gemeinsame Kanten haben.} \\ 0 & \text{sonst.} \end{cases}$

Die Verzögerung, die Paket i erfährt, nennen wir V_i. Nach dem gerade gezeigten Lemma gilt $V_i \leq \sum_{i \neq j} H_{ij}$. Der Ausdruck $\sum_{i \neq j} H_{ij}$ zählt dabei die von ρ_i verschiedenen Routen, die mindestens eine Kante mit ρ_i gemeinsam haben. Jedes H_{ij} ist eine Zufallsvariable mit Wert aus $\{0, 1\}$. Für festes i sind die Zufallsvariablen $H_{i1}, \ldots, H_{iN}$ unabhängig. Wir wollen nun die Wahrscheinlichkeit abschätzen, dass die Verzögerung V_i einen bestimmten Wert überschreitet. Dies machen wir dadurch, dass wir die Chernoffschranke auf die Summe $\sum_{i \neq j} H_{ij}$ anwenden. Wir schätzen dazu als erstes $\mathbf{E}\left[\sum_{i \neq j} H_{ij}\right]$ ab.
Wir fixieren zunächst i und seine Route ρ_i. Für eine Kante e aus dem Hyperwürfel sei $T(e)$ die Zufallsvariable, die misst, für wieviele $j \neq i$ die Route ρ_j die Kante e enthält.
Für eine Route $\rho_i = (e_1, \ldots, e_k)$ gilt $\sum_{i \neq j} H_{ij} \leq \sum_{l=1}^{k} T(e_l)$.

Beobachtung: Sei e eine Kante aus dem Hyperwürfel. Dann gilt: $\mathbf{E}[T(e)] \leq \frac{1}{2}$.

Beweis: Wir betrachten o.B.d.A. eine Kante, entlang der ein 0-Bit zu einem 1-Bit wird:

$$(v_1, \ldots, v_t, 0, v_{t+2}, \ldots, v_n) \rightarrow (v_1, \ldots, v_t, 1, v_{t+2}, \ldots, v_n).$$

Diese Kante wird von einer Route (gemäß bit-fixing) genau dann durchlaufen, wenn der Startort von der Form

$$(*, *, \ldots, *, 0, v_{t+2}, \ldots, v_n)$$

und der Zielort von der Form

$$(v_1, \ldots, v_t, 1, *, *, \ldots, *)$$

ist. (Hierbei steht $*$ für ein beliebiges Bit.) Die Wahrscheinlichkeit, dass ein solcher Zielort als $\sigma(i)$ gewürfelt wird, ist $\left(\frac{1}{2}\right)^{t+1}$. Da es höchstens 2^t viele Startorte gibt, die von der geforderten Form und von i verschieden sind, ist der Erwartungswert $\mathbf{E}[T(e)] = \frac{1}{2}$. □

Also ist $\mathbf{E}[V_i] \leq \mathbf{E}\left[\sum_{i \neq j} H_{ij}\right] \leq \sum_{l=1}^{k} \mathbf{E}[T(e_l)] \leq \frac{k}{2} \leq \frac{n}{2}$. Nun können wir die Chernoffschranke (siehe Abschnitt 3.5) anwenden:

$$\forall R \geq 5\mu : \mathbf{Prob}(V_i \geq R) \leq 2^{-R}.$$

Wir wählen $R = 3n$. Dies ist größer als 5μ. Somit gilt:

$$\mathbf{Prob}(V_i \geq 3n) \leq 2^{-3n}.$$

Da die Anzahl der Pakete 2^n beträgt, ist die Wahrscheinlichkeit, dass mindestens eines der Pakete um mindestens $3n$ Zeiteinheiten verzögert wird, durch 2^{-2n} beschränkt.
Da die Routen eine durch n beschränkte Länge haben, bedeutet eine Verzögerung von höchstens $3n$, dass die Gesamtlaufzeit höchstens $4n$ ist. Zusammenfassend haben wir erhalten:

4.6 Satz. *Mit Wahrscheinlichkeit mindestens* $1 - \left(\frac{1}{2}\right)^{2n}$ *erreicht jedes Paket in der ersten Phase sein Zwischenziel in höchstens* $4n$ *Schritten.*

Die 2. Phase ist analog zur ersten Phase zu analysieren, nur verläuft sie quasi „rückwärts“, da nun die Startorte zufällig sind.
Das einzige Problem, das wir noch haben, ist, dass Pakete, die sich noch in der ersten Phase befinden, Pakete aus der zweiten Phase aufhalten könnten (und umgekehrt). Aus diesem Grund haben wir in den Algorithmus die Wartephase eingebaut. Wenn tatsächlich in der ersten Phase alle Pakete ihr Zwischenziel in maximal $4n$ Schritten erreichen, kann das genannte Problem nicht auftreten.
Wenn A das Ereignis ist, dass nicht alle Pakete in höchstens $4n$ Schritten ihr Zwischenziel erreichen und B das Ereignis, dass nicht alle Pakete höchstens $4n$ Schritte in der zweiten Phase brauchen, dann gilt $\mathbf{Prob}(A \cup B) \leq \mathbf{Prob}(A) + \mathbf{Prob}(B) \leq 2 \cdot \left(\frac{1}{2}\right)^{2n}$, und somit ergibt sich die folgende Aussage:

4.7 Satz. *Die Wahrscheinlichkeit, dass jedes Paket in höchstens* $8n$ *Schritten sein Ziel erreicht, ist mindestens* $1 - 2 \cdot \left(\frac{1}{2}\right)^{2n} \geq 1 - \frac{1}{N}$.

4.2 Randomisiertes Runden

4.2.1 Ein Verdrahtungsproblem („global wiring“)

Gegeben ist ein Gatterarray der Dimensionen $\sqrt{n} \times \sqrt{n}$, bestehend aus n Gattern. Wir nehmen dabei an, dass n eine Quadratzahl ist. Ein Netz ist eine Teilmenge der Gatter, die (z.B. durch einen Draht) miteinander verbunden werden sollen. Diese Verbindungen haben eine Manhattan-Form, die Strecken können also nur horizontal bzw. vertikal verlaufen. Abbildung 4.1 beschreibt ein Beispiel mit $n = 9$, im Array sind drei Netze bestehend aus je zwei Gattern durch Drähte miteinander verbunden.

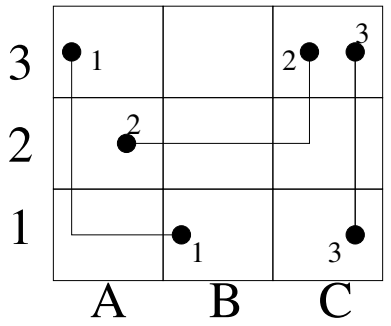


Abbildung 4.1: Ein 3×3-Gatterarray mit drei Netzen.

Das Verdrahtungsproblem besteht in der folgenden Aufgabe: Gegeben ist eine Menge von Netzen, und für jedes Netz sollen alle Gatter des Netzes durch einen Draht miteinander verbunden werden. Dabei sollen gewisse (hardwaretechnisch bedingte) Randbedingungen eingehalten werden. Im folgenden beschränken wir uns auf Netze der Kardinalität zwei.
Das Problem der *globalen* Verdrahtung besteht darin, für jedes Netz die Gatter anzugeben, über die die Drähte verlaufen, im obigen Beispiel führt zum Beispiel der Draht von Netz 3 über die Gatter C3 und C2 zum Gatter C1. Bei der *detaillierten* Verdrahtung wird dann auch noch der Verlauf der Drähte innerhalb der Gatter angegeben.
Wir beschränken uns auf das Problem der globalen Verdrahtung und machen auch noch die Einschränkung, dass die Drähte auf dem Weg vom Startgatter zum Zielgatter nur einmal abbiegen dürfen. Die von uns betrachtete Randbedingung ist die, dass die Anzahl der Drähte, die durch eine horizontale bzw. vertikale Grenze verlaufen, durch einen bestimmten Parameter w beschränkt sein soll. Es gibt zwei Problemvarianten: Die Entscheidungsvariante „Gegeben w: Gibt es eine globale Verdrahtung mit Parameter w?“ sowie die Optimierungsvariante: „Finde das minimale w, so dass es eine globale Verdrahtung mit Parameter w gibt.“ Der optimale Parameter der Optimierungsvariante heiße w_{opt}. Natürlich kann man die Entscheidungsvariante genau dann effizient lösen, wenn man die Optimierungsvariante effizient lösen kann.
Da die Drähte nur einmal abbiegen dürfen, hat man (abgesehen von den Netzen, bei denen Ziel- und Startort in der gleichen Zeile oder Spalte liegen), jeweils genau zwei Wahlmöglichkeiten: Entweder den Startort zunächst horizontal verlassen oder ihn zunächst vertikal verlassen. Wir formulieren das Verdrahtungsproblem nun als mathematisches Programm. Pro Netz führen wir zwei Variablen ein: Zu Netz i gehören die Variablen h_i und v_i. Wenn $(h_i, v_i) = (1, 0)$ ist, dann soll der Startort zunächst horizontal verlassen werden. Falls $(h_i, v_i) = (0, 1)$ ist, dann soll der Startort zunächst vertikal verlassen werden. Für eine Grenze b im Array sei

$$\begin{aligned} T_{b,\text{hor}} &:= \{i \mid \text{Draht zu Netz } i \text{ verläuft durch } b, \text{ wenn } h_i = 1 \text{ ist.}\} \\ T_{b,\text{ver}} &:= \{i \mid \text{Draht zu Netz } i \text{ verläuft durch } b, \text{ wenn } v_i = 1 \text{ ist.}\} \end{aligned}$$

Zum Beispiel sind in Abbildung 4.1 für die Grenze $b = AB3$ die Mengen $T_{b,\text{hor}} = \{1\}$ und $T_{b,\text{ver}} = \{2\}$. (Dabei seien A3, A2 und C1 die Startgatter der Netze.)
Damit können wir das Optimierungsproblem wie folgt beschreiben:

$$\begin{gathered} \textbf{min } w, \\ \text{wobei } \forall i : h_i, v_i \in \{0,1\}, \text{ sowie } h_i + v_i = 1. \\ \text{Und } \forall b : \textstyle\sum_{i \in T_{b,\text{hor}}} h_i + \sum_{i \in T_{b,\text{ver}}} v_i \leq w. \end{gathered}$$

Hierbei legt die dritte Bedingung fest, dass die Anzahl der Drähte durch Grenze b durch w beschränkt sein soll.
Es ist bekannt, dass unser globales Verdrahtungsproblem NP-hart ist, also ist es auch NP-hart, obiges mathematische Programm zu lösen. Wir gehen den Weg der Relaxation, wobei wir die Bedingung, dass h_i und v_i aus $\{0, 1\}$ sein sollen, dahingehend aufweichen, dass wir nur

noch $h_i, v_i \in [0,1]$ fordern. Damit wird aus dem mathematischen Programm ein so genanntes „Lineares Programm“.

Exkurs Lineare Programme

Ein lineares Programm ist gegeben durch eine lineare Zielfunktion $c(x_1, \ldots, x_n) := c_1 x_1 + \cdots + c_n x_n$. Die Aufgabe besteht darin, diese Zielfunktion zu minimieren unter der Randbedingung, dass der Vektor $x_1, \ldots, x_n$ eine Reihe von linearen Bedingungen erfüllt, nämlich

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &\leq d_1 \\ a_{21}x_1 + \cdots + a_{2n}x_n &\leq d_2 \\ \vdots &\quad \vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n &\leq d_m \end{aligned}$$

Die auftretenden Koeffizienten seien dabei rational.
Am Rande sei bemerkt, dass Gleichheitsbedingungen wie zum Beispiel $x = c$ durch zwei Ungleichungen der Form $x \leq c$ und $-x \leq -c$ ausgedrückt werden können.
Es ist bekannt, dass lineare Programme in Polynomialzeit gelöst werden können, genauer, in einer Zeit, die polynomiell in der Eingabelänge des linearen Programms ist (man zählt dort alle Bits mit, die zum Kodieren der einzelnen Koeffizienten benötigt werden.)
Man bekommt als Lösung nicht nur den optimalen Wert der Zielfunktion geliefert, sondern auch die zugehörige Belegung der Variablen $x_1, \ldots, x_n$.

Ende des Exkurses

Das mathematische Programm für unser Verdrahtungsproblem wird nach der Relaxation zu einem linearen Programm, das wir effizient lösen können. Sei nun w_{opt} die optimale Lösung des mathematischen Programms und $w^{(LP)}$ und $h_i^{(LP)}, v_i^{(LP)}$ die optimale Lösung des linearen Programms. (Es gilt also $w^{(LP)} \leq w_{opt}$.) Die Zahlen $h_i^{(LP)}, v_i^{(LP)}$ können nicht-ganzzahlig sein, wie können wir also aus der Lösung eine Verdrahtung bekommen? Der passende Ansatz ist das Randomisierte Runden:

Randomisiertes Runden

Gegeben seien T Variablen $x_1, \ldots, x_T \in [0,1]$. Man interpretiert die x_i als Wahrscheinlichkeiten und erhält $x_1^*, \ldots, x_T^*$ aus den x_i wie folgt:
Setze $x_i^* := 1$ mit Wahrscheinlichkeit x_i und $x_i^* := 0$ mit Wahrscheinlichkeit $1 - x_i$.
Offensichtlich gilt $\mathbf{E}[x_i^*] = x_i$ für alle i.

Deterministisches Runden

Deterministisches Runden definiert die Variablen x_i^* wie folgt: $x_i^* := 1$, falls $x_i \geq 0.5$ ist und $x_i := 0$ sonst.

Hier wenden wir das randomisierte Runden auf die Lösung $h_i^{(LP)}, v_i^{(LP)}$ des linearen Programms an, und zwar setzen wir $(h_i, v_i) := (1,0)$ mit Wahrscheinlichkeit $h_i^{(LP)}$ sowie $(h_i, v_i) := (0,1)$ sonst.

Um einen Satz über die Güte der vom Randomisierten Runden produzierten Verdrahtungen formulieren zu können, benötigen wir die Definition einer Funktion Δ^+:

4.8 Definition. $\Delta^+(\mu, \varepsilon^*)$ sei das kleinste $\delta > 0$, so dass $\left[\frac{e^\delta}{(1+\delta)^{1+\delta}}\right]^\mu \leq \varepsilon^*$ ist.

Den Ausdruck in der Definition sollte man als die rechte Seite der Chernoffschranke wiedererkennen.

4.9 Satz. *Sei $0 < \varepsilon < 1$. Dann gilt: Mit Wahrscheinlichkeit mindestens $1 - \varepsilon$ liefert Randomisiertes Runden mit der Lösung des linearen Programms eine Verdrahtung, deren Parameter w die folgende Ungleichung erfüllt: $w \leq w_{opt} \cdot (1 + \delta)$, für $\delta := \Delta^+(w_{opt}, \varepsilon/2n)$.*

Beweis: Betrachte eine Grenze b. Sei $w(b)$ die Anzahl der Drähte, die durch b verlaufen, wenn wir die Verdrahtung benutzen, die duch die Werte h_i, v_i beschrieben ist. Es gilt $w(b) = \sum_{i \in T_{b,\text{hor}}} h_i + \sum_{i \in T_{b,\text{ver}}} v_i$, somit ist

$$\begin{aligned} \mu := \mathbf{E}[w(b)] &= \sum_{i \in T_{b,\text{hor}}} \mathbf{E}[h_i] + \sum_{i \in T_{b,\text{ver}}} \mathbf{E}[v_i] \\ &= \sum_{i \in T_{b,\text{hor}}} h_i^{(LP)} + \sum_{i \in T_{b,\text{ver}}} v_i^{(LP)} \\ &\leq w^{(LP)} \\ &\leq w_{opt}. \end{aligned}$$

Wir erhalten also insgesamt $\mathbf{E}[w(b)] \leq w_{opt}$. Damit wir aber eine Aussage darüber bekommen, wie wahrscheinlich es ist, dass bei *allen* Grenzen die Anzahl der sie durchlaufenden Drähte in der Nähe von w_{opt} liegt, benötigen wir eine Wahrscheinlichkeitsaussage, die wir mit Hilfe der Chernoffschranken herleiten.
Alle h_i und v_i sind Zufallsvariablen mit Werten aus $\{0, 1\}$. Außerdem sind für $i \neq i'$ die Zufallsvariablen h_i unabhängig von den $h_{i'}$, ebenso bei v. Also ist $w(b)$ eine Summe von unabhängigen Zufallsvariablen, die Werte aus $\{0, 1\}$ annehmen und wir können die Chernoffschranke verwenden: Es gilt:

$$\mathbf{Prob}(w(b) \geq w_{opt} \cdot (1 + \delta)) \leq \left[\frac{e^\delta}{(1+\delta)^{1+\delta}} \right]^{w_{opt}}.$$

Hier haben wir die Variante a0) der Chernoffschranke verwendet.
Für $\delta := \Delta^+(w_{opt}, \varepsilon/2n)$ ist die rechte Seite nicht größer als $\varepsilon/2n$, also ist

$$\mathbf{Prob}(w(b) \geq w_{opt} \cdot (1 + \delta)) \leq \varepsilon/2n.$$

Wieviele Grenzen gibt es im Gatterarray? Genau $2 \cdot (\sqrt{n} - 1) \cdot \sqrt{n} < 2n$ viele. Damit ist die Wahrscheinlichkeit, dass für mindestens eine Grenze b die Eigenschaft $w(b) \geq w_{opt} \cdot (1 + \delta)$ gilt, durch ε beschränkt und somit die Aussage des Satzes gezeigt. □

Wir betrachten nun zwei Fälle, nämlich einmal den Fall, dass der optimale Parameter w_{opt} sehr groß ist und dann den Fall, dass w_{opt} relativ klein ist.

Angenommen, es ist $w_{opt} = n^\gamma$ für ein $\gamma > 0$. Dann wählen wir $\delta := \sqrt{\frac{3 \ln(2n/\varepsilon)}{n^\gamma}}$. Da dies eine Nullfolge ist, ist $\delta = o(1)$ und es gibt also ein n_0, ab dem δ kleiner als 1 ist, so dass wir (ab n_0) die Chernoffschranke in der Version b) verwenden dürfen.
Es ergibt sich, da $\mu \leq w_{opt}$ ist:

$$\begin{aligned} \mathbf{Prob}(w(b) \geq w_{opt} \cdot (1 + \delta)) &\leq e^{-w_{opt} \cdot \delta^2/3} \\ &\leq e^{-n^\gamma \cdot \delta^2/3} \\ &= e^{-\ln(2n/\varepsilon)} \\ &= \varepsilon/2n. \end{aligned}$$

Angenommen, es ist $w_{opt} = O(1)$. Dann ergibt eine Rechnung, dass $\delta = O(\frac{\log n}{\log \log n})$ zu einer Erfolgswahrscheinlichkeit von mindestens $1 - \frac{1}{n}$ führt. Dies ist dann allerdings ein zu schlechtes Ergebnis, denn wir können zeigen:

4.10 Lemma. *Deterministisches Runden liefert eine Verdrahtung mit Parameter $w \leq 2w_{opt}$.*

Beweis: Für alle i gilt $h_i \leq 2h_i^{(LP)}$, denn wenn $h_i^{(LP)} < 0.5$ ist, dann ist $h_i = 0$ und die Aussage gilt trivialerweise, und wenn $h_i^{(LP)} \geq 0.5$ ist, dann ist $h_i = 1$ und $h_i/h_i^{(LP)} \leq 2$. Gleiches gilt natürlich auch für die Variablen v_i. Also gilt

$$\sum_{i \in T_{b,\text{hor}}} h_i^{(LP)} + \sum_{i \in T_{b,\text{ver}}} v_i^{(LP)} \leq w^{(LP)} \quad \Rightarrow \quad \sum_{i \in T_{b,\text{hor}}} h_i + \sum_{i \in T_{b,\text{ver}}} v_i \leq 2w^{(LP)} \leq 2w_{opt}.$$

□

4.2.2 Ein Approximationsalgorithmus für MAXSAT

MAXSAT ist eines der grundlegenden und am meisten untersuchten NP-vollständigen Probleme. Es ist wie folgt beschrieben. Wir haben n Boolesche Variablen $x_1, \ldots, x_n$. Ein Literal ist eine Variable x_i oder eine negierte Variable $\overline{x_i}$. Eine Klausel C ist ein ODER von Literalen. Zum Beispiel ist $(x_1 \vee x_4 \vee \overline{x_6})$ eine Klausel.
Eine Belegung der Variablen ist gegeben durch ein Element $a \in \{0,1\}^n$. Hierbei wird die Variable x_i durch den Wert a_i ersetzt. Eine Belegung a „macht eine Klausel C wahr“ bzw. „erfüllt die Klausel C“, wenn nach Ersetzung der Variablen durch die entsprechenden Werte sich der logische Wert 1 ergibt. Eine Klausel heißt reduziert, wenn sie zu jedem $i \in \{1, \ldots, n\}$ maximal ein Literal auf x_i enthält. Ein Beispiel: Die Klauseln $(x_1 \vee x_1 \vee x_5)$ bzw. $(x_1 \vee \overline{x_1} \vee x_2)$ sind nicht reduziert, $(x_1 \vee \overline{x_3} \vee x_2)$ ist reduziert.

MAXSAT

Gegeben:	Eine Liste von reduzierten Klauseln.
Gesucht:	Eine Belegung der Variablen, die möglichst viele Klauseln gleichzeitig erfüllt.

MAX-k-SAT ist die Variante von MAXSAT, bei der alle Klauseln Länge genau k haben. MAX-SAT ist ein NP-hartes Problem, und auch MAX-k-SAT ist NP-hart für jedes $k \geq 2$.

Ein randomisierter Algorithmus

Wir schauen uns zunächst einen einfachen randomisierten Algorithmus an.

4.11 Satz. *Wenn man die Variablenbelegung durch randomisiertes Runden mit den Parametern $(1/2, \ldots, 1/2)$ berechnet, so gilt:*
Bei m reduzierten Klauseln $C_1, \ldots, C_m$ der Länge k erfüllt die berechnete Variablenbelegung im Erwartungswert $(1 - \frac{1}{2^k}) \cdot m$ Klauseln. Die Laufzeit ist $O(n)$.

Beweis: Der Algorithmus setzt der Reihe nach, jeweils mit Wahrscheinlichkeit 1/2 und unabhängig voneinander, die Variablen auf 1 bzw. 0. Für die i-te Klausel C_i definieren wir die Zufallsvariable X_i wie folgt:

$$X_i := \begin{cases} 1, & \text{falls die } i\text{-te Klausel erfüllt ist.} \\ 0 & \text{sonst.} \end{cases}$$

Es gilt $\mathbf{E}[X_i] = \mathbf{Prob}(\text{die } i\text{-te Klausel ist erfüllt})$. Sei $X = X_1 + \cdots + X_m$ die Zufallsvariable, die angibt, wieviele der Klauseln erfüllt sind. Wegen der Linearität des Erwartungswerts ist $\mathbf{E}[X] = \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_m]$.
Eine reduzierte Klausel mit k Literalen ist für genau eine der 2^k möglichen Belegungen ihrer Literale nicht erfüllt. Die Wahrscheinlichkeit, dass sie bei zufälliger Variablenbelegung erfüllt

ist, ist also $1-(\frac{1}{2})^k$. Somit ist $\mathbf{E}[X_i] = 1-(\frac{1}{2})^k$. Also ist $\mathbf{E}[X] = (1-\frac{1}{2^k})\cdot m$. □

4.2.3 Ein Approximationsalgorithmus für MAXSAT mit Güte 0.75

Eine äquivalente Formulierung für das MAXSAT-Problem ist die folgende:

$$\textbf{max } \sum_{j=1}^{m} z_j \text{ unter den Nebenbedingungen}$$

$$y_i, z_j \in \{0,1\} \text{ für alle } i = 1,\ldots,n \text{ und } j = 1,\ldots,m.$$

$$\sum_{x_i \text{ kommt in Klausel } j \text{ vor}} y_i + \sum_{\overline{x_i} \text{ kommt in Klausel } j \text{ vor}} (1-y_i) \geq z_j \text{ für } j = 1,\ldots,m.$$

Dabei zählt der Ausdruck

$$\sum_{x_i \text{ kommt in Klausel } j \text{ vor}} y_i + \sum_{\overline{x_i} \text{ kommt in Klausel } j \text{ vor}} (1-y_i)$$

genau, wieviele Literale in der j-ten Klausel wahr sind, wenn man die Variablen $x_i := y_i$ setzt. Da der Ausdruck $\sum_{j=1}^{m} z_j$ maximiert wird, gibt z_j dann an, ob die j-te Klausel erfüllt ist (also $z_j = 1$) oder nicht (also $z_j = 0$).
Da MAXSAT NP-hart ist, ist es auch NP-hart, das obige „mathematische Programm“ zu lösen. Wir relaxieren das gegebene Programm, indem wir die Bedingung $y_i, z_j \in \{0,1\}$ durch die Bedingung $y_i, z_j \in [0,1]$ ersetzen.
Dadurch wird aus dem gegebenen Problem ein lineares Programm. Da wir relaxiert haben, gilt, dass der optimale Zielfunktionswert des linearen Programms mindestens so groß ist wie die maximal erfüllbare Anzahl der Klauseln im MAXSAT-Problem. Als Beispiel, dass das lineare Programm einen optimalen Zielfunktionswert haben kann, der größer ist als die Anzahl erfüllbarer Klauseln, betrachten wir die folgenden vier Klauseln $(x_1 \vee x_2), (x_1 \vee \bar{x}_2), (\bar{x}_1 \vee x_2), (\bar{x}_1 \vee \bar{x}_2)$, Natürlich sind nur 3 Klauseln gleichzeitig erfüllbar, aber wenn wir im zugehörigen Programm alle Variablen auf $1/2$ setzen, so können alle z_i-Variablen den Wert 1 annehmen und der optimale Zielfunktionswert beim linearen Programm ist 4.
Übrigens sieht man damit, dass es keinen Sinn macht, das Lineare Programm zu lösen, wenn man eine MAXSAT-Instanz vorliegen hat, in der alle Klauseln die Länge mindestens 2 haben, denn dann kennt man eine optimale Lösung auch so: es reicht, alle y-Variablen auf $1/2$ zu setzen.
Wir nehmen nun an, dass wir zum linearen Programm die Lösungen $\widehat{y_i}$ und $\widehat{z_j}$ gegeben haben. Diese Werte liegen alle zwischen 0 und 1. Wie bekommen wir daraus eine Belegung der Variablen? Wir wählen als Vorgehen wieder das „Randomisierte Runden“:

for $i = 1$ **to** n **do**
begin
  setze Variable x_i mit Wahrscheinlichkeit $\widehat{y_i}$ auf 1, (und 0 sonst).
end

Betrachten wir ein Beispiel. Wenn die folgenden drei Klauseln gegeben sind: $C_1 = x_1 \vee \overline{x_2}, C_2 = \overline{x_1} \vee \overline{x_2}, C_3 = x_2$, dann überzeugen wir uns leicht, dass maximal zwei der drei Klauseln gleichzeitig zu erfüllen sind, und dass dies zum Beispiel durch die Belegung $x_1 = 1, x_2 = 1$ geschieht.
Als lineares Programm erhalten wir: **max** $z_1 + z_2 + z_3$ unter den Nebenbedingungen

$$\begin{aligned} x_1 + (1-x_2) &\geq z_1 \\ (1-x_1) + (1-x_2) &\geq z_2 \\ x_2 &\geq z_3 \\ \forall i = 1,\ldots,3: \quad & 0 \leq x_i, z_i \leq 1. \end{aligned}$$

Wenn man dieses Programm löst, erhält man, dass das Optimum für (z.B.)

$$x_1 = 1/2, x_2 = 1/2, z_1 = 1, z_2 = 1, z_3 = 1/2$$

angenommen wird, der Zielfunktionswert ist 2.5. Randomisiertes Runden mit den Parametern $1/2, 1/2$ erfüllt die erste Klausel mit Wahrscheinlichkeit $3/4$, die zweite mit Wahrscheinlichkeit $3/4$, die dritte mit Wahrscheinlichkeit $1/2$ und wir sehen, dass der Erwartungswert für die Anzahl der erfüllten Klauseln bei diesem Randomisierten Runden genau 2 ist.

Wir analysieren nun allgemein, mit welcher Wahrscheinlichkeit die j-te Klausel erfüllt wird.

4.12 Lemma („Klausellemma“)**.** *Sei die j-te Klausel eine Klausel der Länge k. Die Wahrscheinlichkeit, dass sie durch Randomisiertes Runden erfüllt wird, ist mindestens*

$$[1 - (1 - \frac{1}{k})^k] \cdot \widehat{z_j}.$$

Vor dem Beweis des Lemmas zwei einfache Eigenschaften. Die erste entspricht der Aussage „Das geometrische Mittel ist nicht größer als das arithmetische Mittel“ und die zweite zeigen wir im Anhang.

Lemma 1: Seien k Zahlen $a_1, \ldots, a_k \geq 0$ gegeben. Dann gilt

$$a_1 \cdots a_k \leq (\frac{a_1 + \cdots + a_k}{k})^k.$$

Lemma 2: Auf dem Intervall $x \in [0, 1]$ gilt $1 - (1 - \frac{x}{k})^k \ \geq \ (1 - (1 - \frac{1}{k})^k) \cdot x$.

Beweis: (**„Klausellemma“**)
Sei die Klausel o.B.d.A. durch $x_1 \vee \cdots \vee x_k$ gegeben. Die Klausel ist dann mit Wahrscheinlichkeit $\prod_{i=1}^k (1 - \widehat{y_i})$ nicht erfüllt, also mit Wahrscheinlichkeit $1 - \prod_{i=1}^k (1 - \widehat{y_i})$ erfüllt.
Es ist $\widehat{y_1} + \cdots + \widehat{y_k} \geq \widehat{z_j}$, also

$$(1 - \widehat{y_1}) + \cdots + (1 - \widehat{y_k}) \leq k - \widehat{z_j},$$

also nach Lemma 1

$$\prod_{i=1}^k (1 - \widehat{y_i}) \leq (\frac{k - \widehat{z_j}}{k})^k, \text{ also } 1 - \prod_{i=1}^k (1 - \widehat{y_i}) \geq 1 - (1 - \frac{\widehat{z_j}}{k})^k.$$

Nach Lemma 2 ist dies mindestens $(1 - (1 - \frac{1}{k})^k) \cdot \widehat{z_j}$. Man beachte, dass für alle k gilt, dass $1 - (1 - \frac{1}{k})^k \geq 1 - \frac{1}{e} \approx 0.632$ ist.

□

4.13 Satz. *Gegeben sei eine Menge von Klauseln, von denen maximal OPT gleichzeitig erfüllt werden können. Sei $\widehat{y_1}, \ldots, \widehat{y_n}$ eine optimale Lösung des linearen Programms.*
Randomisiertes Runden mit den Parametern $\widehat{y_1}, \ldots, \widehat{y_n}$ liefert eine Variablenbelegung, die im Erwartungswert mindestens $(1 - \frac{1}{e}) \cdot OPT$ Klauseln erfüllt.

Beweis: Da das lineare Programm eine Relaxation ist, gilt $\text{OPT} \leq \sum_{j=1}^m \widehat{z_j}$. Wenn Klausel Nummer j die Länge k_j besitzt, dann folgt aus dem Klausellemma, dass die erwartete Anzahl erfüllter Klauseln mindestens

$$\begin{aligned} \sum_j [1 - (1 - \frac{1}{k_j})^{k_j}] \cdot \widehat{z_j} &\geq (1 - (1/e)) \cdot \sum \widehat{z_j} \\ &\geq (1 - (1/e)) \cdot \text{OPT} \end{aligned}$$

ist. □

Nun kommen wir zur Beschreibung eines Verfahrens, das Approximationsgüte 0.75 hat.

Verfahren MIX: Berechne eine Variablenbelegung a durch Randomisiertes Runden mit den Parametern $1/2, \ldots, 1/2$. Berechne eine Variablenbelegung b durch Randomisiertes Runden mit den Parametern $\hat{y}_1, \ldots, \hat{y}_n$, die sich aus der optimalen Lösung des linearen Programms ergeben. Gib diejenige Variablenbelegung von a und b aus, die mehr Klauseln erfüllt.

4.14 Satz. *Sei eine Menge von Klauseln gegeben und sei OPT die Anzahl an Klauseln, die maximal gleichzeitig erfüllt werden kann. Dann liefert das Verfahren MIX eine Variablenbelegung, die im Erwartungswert mindestens* $0.75 \cdot OPT$ *viele Klauseln erfüllt.*

Beweis: In unseren obigen Analysen haben wir als Nebenprodukt erhalten, dass bei Randomisiertem Runden mit Parametern $1/2, \ldots, 1/2$ eine Klausel der Länge k mit Wahrscheinlichkeit $A(k) := 1 - 2^{-k}$ erfüllt ist und bei Randomisiertem Runden mit den Parametern $\hat{y}_1, \ldots, \hat{y}_n$ mit Wahrscheinlichkeit $B(k) \cdot \hat{z}_j$ erfüllt ist, wenn wir $B(k) := 1 - (1 - \frac{1}{k})^k$ setzen. Wir verschaffen uns eine kleine Übersicht über die Werte $A(k)$ und $B(k)$ für einige Werte von k:

k	$A(k)$ $1 - 2^{-k}$	$B(k)$ $1 - (1 - \frac{1}{k})^k$	$(A(k) + B(k))/2$
1	0.5	1.0	0.75
2	0.75	0.75	0.75
3	0.875	0.704	0.7895
4	0.938	0.684	0.811
5	0.969	0.672	0.8205

Sei n_1 die Anzahl an Klauseln, die durch Randomisiertes Runden mit $(1/2, \ldots, 1/2)$ erfüllt werden, und sei n_2 die Anzahl an Klauseln, die durch Randomisiertes Runden mit Parametern $\hat{y}_1, \ldots, \hat{y}_n$ erfüllt werden. Der Algorithmus MIX erfüllt $\max\{n_1, n_2\}$ Klauseln.
Wir zeigen, dass $\mathbf{E}[\max\{n_1, n_2\}] \geq \frac{3}{4} \cdot \sum_j \hat{z}_j$ ist, denn dann folgt $\mathbf{E}[\max\{n_1, n_2\}] \geq \frac{3}{4} \cdot \mathrm{OPT}$.
Da $\max\{n_1, n_2\} \geq \frac{n_1+n_2}{2}$ ist („das Maximum zweier Zahlen ist mindestens so groß wie der Durchschnitt der beiden"), reicht es, zu zeigen, dass

$$\mathbf{E}\left[\frac{n_1 + n_2}{2}\right] \geq \frac{3}{4} \cdot \sum_j \hat{z}_j \text{ ist.}$$

Sei $\ell(C)$ die Länge einer Klausel C. Es ist

$$\begin{aligned} \mathbf{E}[n_1] &= \sum_{j=1}^{m} A(\ell(C_j)) \geq \sum_{j=1}^{m} A(\ell(C_j)) \cdot \hat{z}_j \text{ sowie} \\ \mathbf{E}[n_2] &= \sum_{j=1}^{m} B(\ell(C_j)) \cdot \hat{z}_j. \end{aligned}$$

Somit ist

$$\mathbf{E}[n_1] + \mathbf{E}[n_2] \geq \sum_{j=1}^{m} (A(\ell(C_j)) + B(\ell(C_j))) \cdot \hat{z}_j.$$

Eine einfache Rechnung zeigt, dass $A(k) + B(k) \geq 3/2$ für alle natürlichen Zahlen k ist:
Für $k = 1$ und $k = 2$ ergibt die Summe den Wert $3/2$, für $k \geq 3$ beobachten wir $(1 - 2^{-k}) \geq 7/8$ und $1 - (1 - \frac{1}{k})^k \geq 1 - (1/e)$, damit ist die Summe mindestens

$$(7/8) + 1 - (1/e) \approx 1.507\ldots \geq 3/2.$$

Damit folgt die Aussage. □

Anhang:

4.15 Behauptung. Sei $I = [a, b]$ ein Intervall. Wenn eine Funktion f differenzierbar auf I ist und f' stetig und monoton fallend auf I ist, dann gilt folgendes:

$$f(a) = f(b) \Rightarrow f(x) \geq f(a) \text{ für alle } x \in [a, b].$$

Beweis: Angenommen, es gäbe ein $x^* \in [a, b]$ mit $f(x^*) < f(a)$. Nach dem Zwischenwertsatz gibt es dann ein $x^{**} \in [a, x^*]$ mit $f'(x^{**}) = \frac{f(x^*)-f(a)}{x^*-a} < 0$. Da f' monoton fallend ist, ist f' negativ auf dem Intervall $[x^*, b]$, also ist f monoton fallend auf $[x^*, b]$ und somit $f(b) \leq f(x^*) < f(a)$, was ein Widerspruch zur Voraussetzung $f(a) = f(b)$ ist. □

Damit erhalten wir:

4.16 Lemma. *Es ist* $1 - \left(1 - \frac{x}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot x$ *für alle* $x \in [0, 1]$ *und* $k \geq 1$.

Beweis: Sei $f(x) := 1 - \left(1 - \frac{x}{k}\right)^k - cx$ mit $c := 1 - \left(1 - \frac{1}{k}\right)^k$.
Dann sind die Voraussetzungen des letzten Lemmas erfüllt für das Intervall $[a, b] = [0, 1]$, denn es ist $f(0) = 0 = f(1)$ und $f'(x) = \left(1 - \frac{x}{k}\right)^{k-1} - c$ ist monoton fallend auf $[0, 1]$. Also ist $f(x) \geq 0$ für alle $x \in [0, 1]$ und somit gilt das Lemma. □

4.2.4 Das Problem Hitting Set

Dieses Problem kann, wie wir zeigen, zwar mit Randomisiertem Runden gelöst werden, aber ein einfacher Greedyalgorithmus kann ebenso benutzt werden.

Problem Hitting Set
Eingabe: Mengen $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$.
Ausgabe: Eine Menge T heißt Hitting Set für $S_1, \ldots, S_m$, wenn $T \cap S_i \neq \emptyset$ für alle $i = 1, \ldots, m$ ist. Gesucht ist ein Hitting Set T mit minimaler Kardinalität $|T|$. Dieses Problem ist NP-hart.

4.17 Beispiel. Die Menge $T = \{1, \ldots, n\}$ ist immer ein Hitting Set.
Wenn $S_1 = \{1, 3, 5, 7\}, S_2 = \{2, 4, 5, 8\}, S_3 = \{3, 4, 6, 9\}$ ist, dann ist $T = \{1, 4\}$ ein Hitting Set.

4.18 Behauptung. Gegeben sei eine Eingabe für das Hitting Set Problem. Mit randomisiertem Runden kann man einen Hitting Set T finden mit $|T| \leq (2 \ln m + O(1)) \cdot c_{opt}$, wenn c_{opt} die minimale Kardinalität eines Hitting Sets für die gegebene Eingabe bezeichnet.

Beweis: Wir stellen folgendes mathematische Programm auf, das äquivalent zum Hitting-Set-Problem ist.
min $\sum_{i=1}^{n} x_i$,
unter den NB $x_i \in \{0, 1\}$ für alle $i = 1, \ldots, n$.
$\forall j \in \{1, \ldots, m\} : \sum_{i \in S_j} x_i \geq 1$.

In diesem mathematischen Programm dienen die x_i als Indikatorvariablen, die anzeigen, ob Element i in die Menge T gewählt wird oder nicht. Die zweite Nebenbedingung formuliert, dass jede Menge S_j „abgedeckt“ ist.
Wir relaxieren dieses Programm zu einem linearen Programm, indem wir $x_i \in [0, 1]$ zulassen.

Die optimale Lösung des linearen Programms sei gegeben durch den Zielfunktionswert $c^{(LP)}$ sowie die Variablenbelegungen $x_1^{(LP)}, \ldots, x_n^{(LP)}$. Es gilt also $c^{(LP)} = x_1^{(LP)} + \cdots + x_n^{(LP)}$. Da das lineare Programm eine Relaxation des Hitting Set Problems ist, gilt $c^{(LP)} \leq c_{opt}$.
Wir führen randomisiertes Runden durch und wählen für jedes $i = 1, \ldots, n$ Element i in die Menge T mit Wahrscheinlichkeit $x_i^{(LP)}$. Damit gilt $\mathbf{E}[|T|] = c^{(LP)}$.
Nun analysieren wir die Wahrscheinlichkeit, dass Menge S_j nicht abgedeckt ist, dass also $S_j \cap T = \emptyset$ gilt. Wir nehmen o.B.d.A. an, dass $S_j = \{1, \ldots, k\}$ gilt, dann ist

$$\mathbf{Prob}(S_j \cap T = \emptyset) = (1 - x_1^{(LP)}) \cdots (1 - x_k^{(LP)}).$$

Da wir schon bei dem MAXSAT-Problem eine ähnliche Rechnung durchgeführt haben, wissen wir, dass dieser Ausdruck (da $\sum_{i=1}^{k} x_i^{(LP)} \geq 1$ ist) kleiner oder gleich $1/e$ ist.
Wir wissen also, dass für alle $j = 1, \ldots, m$ gilt, dass $\mathbf{Prob}(S_j \cap T = \emptyset) \leq 1/e$ ist.
Bei der von uns ausgewählten Menge T können wir weder garantieren, dass sie ein Hitting Set ist, noch können wir garantieren, dass sie eine bestimmte Kardinalität hat.
Wir iterieren das obige randomisierte Runden t mal und erhalten t Mengen $T_1, \ldots, T_t$. Als Menge T wählen wir nun $T = T_1 \cup T_2 \cup \cdots \cup T_t$.
Die Zahl t sollte groß genug sein, damit wir mit großer Wahrscheinlichkeit alle Mengen S_j abgedeckt haben. Wie groß wir t wählen, wird nun eine Rechnung zeigen. Es gilt:

$$\mathbf{E}[|T|] = \mathbf{E}[|T_1 \cup \cdots \cup T_t|] \leq \mathbf{E}[|T_1| + \cdots + |T_t|] = t \cdot c^{(LP)} \leq t \cdot c_{opt}$$

und $\mathbf{Prob}(S_j \cap T = \emptyset) \leq (1/e)^t$.
Wir definieren nun, dass unser Algorithmus Pech gehabt hat, wenn $|T| \geq 2 \cdot t \cdot c_{opt}$ ist oder wenn es eine Menge S_j gibt, für die $S_j \cap T = \emptyset$ gilt. Es gilt

$$\begin{aligned} \mathbf{Prob}(\text{Algorithmus hat Pech gehabt}) &\leq \mathbf{Prob}(|T| \geq 2 \cdot t \cdot c_{opt}) + \mathbf{Prob}(\exists j : S_j \cap T = \emptyset) \\ &\leq 1/2 + m \cdot (1/e)^t. \end{aligned}$$

Dabei haben wir für den ersten Term die Markovungleichung verwendet.
Damit diese Wahrscheinlichkeit kleiner gleich $3/4$ wird, wählen wir $t \geq \ln m + \ln 4$.
Wir haben also einen Algorithmus beschrieben, der mit Erfolgswahrscheinlichkeit mindestens $1/4$ eine Menge T mit $|T| \leq 2(\ln m + O(1)) \cdot c_{opt}$ berechnet, die ein Hitting Set ist. □

Wir zeigen nun, dass ein Greedyalgorithmus allerdings besser ist:

Greedyalgorithmus:
$T := \emptyset; M := \{1, \ldots, m\}$.
# T wird zu einem Hitting Set aufgebaut,
# M ist die Menge der Indizes $\{j \mid S_j \cap T = \emptyset\}$.
while $M \neq \emptyset$ **do**
begin
  wähle $x \in \{1, \ldots, n\} \setminus T$, das die meisten S_j (mit $j \in M$) abdeckt.
  $T := T \cup \{x\}$. Aktualisiere M.
  (Für die folgende Analyse sei M_{neu} die neue Menge M.)
end;

4.19 Behauptung. Der Greedyalgorithmus berechnet einen Hitting Set T der Kardinalität $|T| \leq \lceil \ln m \cdot c_{opt} \rceil$, wenn c_{opt} die Kardinalität eines minimalen Hitting Sets ist. Seine Güte ist also durch $\ln m$ beschränkt.

Beweis: Da $|T|$ die Anzahl der Durchläufe durch die while-Schleife ist, reicht es, diese abzuschätzen. Zu diesem Zweck betrachten wir den Verlauf von $|M|$ im Laufe des Algorithmus.
Nach 0 Durchläufen der while-Schleife ist $|M| = m$. Da zu Beginn c_{opt} Elemente ausreichen, um alle Mengen abzudecken, reichen auch für die kleineren Probleme, die im Laufe des Algorithmus

entstehen, c_{opt} Elemente. Damit gibt es stets ein Element, das mindestens $|M|/c_{opt}$ viele Mengen abdeckt. Also gilt $|M_{neu}| \leq |M| \cdot (1 - \frac{1}{c_{opt}})$. Nach $i \geq 1$ Durchläufen der while-Schleife gilt somit für die dann aktuelle Menge M, dass $|M| \leq m \cdot (1 - \frac{1}{c_{opt}})^i < m \cdot e^{-i/c_{opt}}$ ist. Damit ist klar, dass nach spätestens $\lceil c_{opt} \cdot \ln m \rceil$ Schleifendurchläufen die Menge M die leere Menge ist. □

4.3 Ein 0.878-Approximationsalgorithmus für MAXCUT

Bevor wir diesen beschreiben können, machen wir einen kleinen Exkurs in die Matrizentheorie bzw. die Lineare Algebra.

4.3.1 Exkurs Matrizen/lineare Algebra

Wir benötigen ein paar Begriffe aus der Matrixtheorie bzw. der linearen Algebra. Diese kann man z.B. auch in einem Buch von G.H. Golub und C.F. van Loan nachlesen: *Matrix Computations, North Oxford Academic, 1986.*

4.3.2 Grundlagen der Matrixtheorie

Wir gehen in diesem Kapitel davon aus, dass die betrachteten Matrizen nur Einträge aus den reellen Zahlen haben (also nicht etwa komplexe Zahlen).
Eine $n \times m$-Matrix heißt *quadratisch*, wenn $n = m$ ist. Eine Matrix A mit $A = A^T$ heißt *symmetrisch*. Id bezeichnet in diesem Kapitel die Identitätsmatrix.

4.20 Definition. Ein *Eigenvektor* einer Matrix A ist ein Vektor $x \neq 0$, für den eine komplexe Zahl λ existiert mit $A \cdot x = \lambda \cdot x$. Der Wert λ heißt *Eigenwert*.

Eigenwerte lassen sich auch noch anders charakterisieren:

4.21 Lemma. *Die Eigenwerte einer Matrix A sind die Nullstellen des Polynoms* $\det(A - \lambda \cdot Id)$*, wobei λ die Variable des Polynoms ist.*

Für symmetrische $n \times n$-Matrizen A ist folgendes bekannt: A besitzt n reelle Eigenwerte (die nicht notwendigerweise alle voneinander verschieden sind.) Diese n Eigenwerte werden häufig geschrieben als $\lambda_1, \ldots, \lambda_n$, wobei man eine kanonische Numerierung annimmt, so dass $\lambda_1 \geq \ldots \geq \lambda_n$ gilt.
Wenn ein Eigenwert mehrmals vorkommt, dann wird die Anzahl seines Vorkommens auch *Multiplizität* genannt.

4.22 Satz (Rayleigh-Ritz)**.** *Wenn A eine symmetrische Matrix ist, dann gilt*

$$\lambda_1(A) = \max_{||x||=1} x^T \cdot A \cdot x \qquad \textit{und} \qquad \lambda_n(A) = \min_{||x||=1} x^T \cdot A \cdot x.$$

4.23 Definition. Eine quadratische Matrix A heißt

$$\left.\begin{array}{lll} \textit{positiv semidefinit,} & \text{wenn} & x^T \cdot A \cdot x \geq 0 \\ \textit{positiv definit,} & \text{wenn} & x^T \cdot A \cdot x > 0 \end{array}\right\} \text{für alle } x \in \mathbf{R}^n \setminus \{\underline{0}\} \text{ ist.}$$

In diesem Skript werden wir den Begriff „positiv semidefinit“ auch abkürzen als „PSD“.

Als Korollar des Satzes von Rayleigh-Ritz erhalten wir, dass eine symmetrische Matrix PSD ist genau dann, wenn ihr kleinster Eigenwert $\lambda_n \geq 0$ ist und positiv definit, wenn $\lambda_n > 0$ ist.
Wir wollen nun einige Beispiele für Matrizen bringen, die PSD sind. Zunächst sollten wir festhalten, dass eine Matrix, die nur positive Einträge hat, nicht unbedingt PSD ist: Wir betrachten die folgenden beiden Matrizen:

$$A := \begin{pmatrix} 1 & 4 \\ 4 & 1 \end{pmatrix} \qquad B := \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix}.$$

Die Matrix A ist nicht PSD. Dies sieht man zum Beispiel daran, dass $(1,-1) \cdot A \cdot (1,-1)^T = -6$ ist. Wenn man die Eigenwerte von A ausrechnet, findet man $\lambda_1 = 5, \lambda_2 = -3$.
Es gilt zum Beispiel $A \cdot (1,1)^T = (5,5)^T$ und $A \cdot (1,-1)^T = (-3,3)^T$.
Die Matrix B enthält negative Einträge, ist aber „trotzdem“ PSD. Dies liegt daran, dass die beiden Eigenwerte von B ungefähr die Werte 4.30 und 0.697 haben, also beide echt größer als Null sind.
Für eine Diagonalmatrix sind die Eigenwerte identisch mit den Einträgen auf der Diagonale. Eine Diagonalmatrix ist also genau dann PSD, wenn alle ihre Diagonaleinträge nicht kleiner als Null sind.
Aus der Definition der Semidefinitheit folgt auch, dass folgendes gilt: Wenn B und C Matrizen sind, dann ist die Matrix

$$A = \begin{pmatrix} B & 0 \\ 0 & C \end{pmatrix}$$

PSD genau dann, wenn B und C beide PSD sind. Dies bedeutet, dass man eine Bedingung der Form „die Matrizen $A_1, \ldots, A_r$ sollen PSD sein“ auch ausdrücken kann als „die Matrix A^* soll PSD sein“, wenn man die Matrix A^* analog zu gerade aus den A_i zusammensetzt.
In diesem Kapitel haben wir meistens mit symmetrischen Matrizen zu tun. Für symmetrische Matrizen gilt die folgende Äquivalenzaussage:

4.24 Lemma. *Sei A eine symmetrische $n \times n$-Matrix. Die folgenden Aussagen sind äquivalent:*

- *A ist PSD.*
- *Alle Eigenwerte von A sind nicht-negativ.*
- *Es gibt eine $n \times n$-Matrix B, so dass man A schreiben kann als $A = B^T \cdot B$.*

Die Zerlegung $A = B^T \cdot B$ kann man auch anders lesen: Es gibt n Vektoren $b_1, \ldots, b_n$, so dass jedes Element $A_{i,j} = b_i^T \cdot b_j$ ist. Die Vektoren b_i entsprechen den Spalten in B.
Diese Zerlegung bezeichnet man auch als „(unvollständige) Cholesky-Zerlegung“. Es gibt einen Algorithmus, der eine solche Cholesky-Zerlegung in Laufzeit $O(n^3)$ berechnet, man lese zum Beispiel im früher erwähnten Buch von Golub und van Loan nach.
Der erwähnte Algorithmus bemerkt auch, wenn die vorgelegte symmetrische Matrix *nicht* PSD ist.
Wen es nun stört, dass wir zwar für symmetrische Matrizen entscheiden können, ob sie PSD sind oder nicht, aber anscheinend nicht für beliebige Matrizen, der möge durch folgende Überlegung beruhigt werden:
Das Problem, zu entscheiden, ob eine gegebene quadratische Matrix A PSD ist, kann zurückgeführt werden auf das gleiche Problem für eine symmetrische Matrix A'. Wir können nämlich A' wie folgt definieren: $A'_{i,j} := \frac{1}{2} \cdot (A_{i,j} + A_{j,i})$. Da

$$x^T \cdot A \cdot x \;=\; \sum_{i,j} A_{i,j} \cdot x_i \cdot x_j$$

ist, gilt, dass $x^T \cdot A' \cdot x = x^T \cdot A \cdot x$ ist und somit ist die symmetrische Matrix A' genau dann PSD, wenn die Matrix A PSD ist.

Wenn wir eine symmetrische Matrix A haben, die PSD ist, dann ist die Zerlegung $A = B^T \cdot B$ nicht eindeutig, weil das Skalarprodukt von Vektoren rotationsinvariant ist, also liefert die Rotation der Vektoren, die durch die Matrix B beschrieben werden, eine neue Zerlegung. Insbesondere kann man B immer so wählen, dass B eine obere (oder untere) Dreiecksmatrix ist.
Die folgende Matrix A ist positiv definit, wir geben auch eine Zerlegung der Form $A = B^T \cdot B$ an:

$$A = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & \sqrt{3} \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 \\ 0 & \sqrt{3} \end{pmatrix}.$$

4.3.3 Semidefinite Programmierung

In der Informatik sind viele wichtige Probleme *NP*-hart und somit vermutlich nicht auf effiziente Weise lösbar. Oft gibt man sich für solche Probleme dann mit Algorithmen zufrieden, die nicht die optimale Lösung finden, sondern nur eine „gute“ Lösung. Ein Approximationsalgorithmus garantiert, dass die gefundene Lösung nicht viel schlechter ist als die optimale Lösung. Angenommen, wir haben ein Maximierungsproblem. Ein 1/2-Approximationsalgorithmus für dieses Problem ist dann z.B. ein Algorithmus, der Lösungen abliefert, die mindestens halb so groß sind wie das Optimum. In den letzten Jahren (seit 1994) hat es eine interessante Entwicklung gegeben. Es ist den Forschern und Forscherinnen gelungen, bessere Approximationsalgorithmen zu entwerfen, und zwar für eine Reihe von Problemen. Diese Algorithmen basieren auf einer Verallgemeinerung der Linearen Programmierung. Es handelt sich um die sogenannte Semidefinite Programmierung. Wir wollen hier einen kurzen Eindruck geben, worum es sich bei der Semidefiniten Programmierung handelt und nur erwähnen, dass es Polynomialzeitalgorithmen gibt, die Probleme der Semidefiniten Programmierung lösen können. Diese Algorithmen basieren auf einer Verallgemeinerung von „Innere-Punkt-Methoden“, die ursprünglich für die Lineare Programmierung entwickelt worden sind.
Um ein Beispiel zu betrachten, wie man die Semidefinite Programmierung für in der Informatik auftretende Probleme verwenden kann, schauen wir uns auch das Beispiel an, das gewissermaßen die Forschungslawine in den letzten Jahren ausgelöst hat, nämlich das Problem MAXCUT.

4.3.4 Semidefinite Programme

In der Literatur gibt es verschiedene Definitionen von Semidefiniten Programmen. Wie man jedoch nicht anders erwarten sollte, sind diese Definitionen (oft oder immer?) äquivalent. Wir wählen die folgende Definition:

4.25 Definition. Ein *semidefinites Programm* hat die folgende Form: Wir suchen eine Lösung in den reellen Variablen $x_1, \ldots, x_m$. Gegeben ist ein Vektor $c \in \mathbf{R}^m$, und wir wollen die Zielfunktion $c^T \cdot x$ minimieren (oder maximieren). Die zulässige Menge ist beschrieben durch eine symmetrische Matrix *SP*, die als Einträge lineare Funktionen in den x_i-Variablen enthält. Genauer gesagt: Ein Vektor a ist zulässig, wenn die Matrix *SP* PSD wird sobald wir die Variablen x_i in *SP* mit den Werten aus a belegen, also $x_i = a_i$ setzen.

Hier ist ein Beispiel eines semidefiniten Programms:

$$\begin{array}{rl} \textbf{max} & x_1 + x_2 + x_3 \\ \textbf{so dass} & \begin{pmatrix} 1 & x_1 - 1 & 2x_1 + x_2 - 1 \\ x_1 - 1 & -x_1 & x_2 + 3 \\ 2x_1 + x_2 - 1 & x_2 + 3 & 0 \end{pmatrix} \text{ PSD ist.} \end{array}$$

Wenn wir ein Problem als semidefinites Programm modellieren möchten, muß sich nicht unbedingt eine symmetrische Matrix ergeben. Entsprechend der vorhin gemachten Bemerkungen kann man „unsymmetrische Bedingungen“ durch symmetrische Bedingungen ausdrücken.

Die Matrix, die wir erhalten, wenn wir die Variablen x_i gemäß dem Vektor a belegen und in *SP* einsetzen, bezeichnen wir hier mit $SP(a)$.
Wir bemerken, dass es Methoden gibt, ein semidefinites Programm zu lösen. Genauer: Zu jedem gegebenen $\varepsilon > 0$ und jedem semidefiniten Programm, das eine endliche, polynomiell beschränkte Lösung hat, kann man das Programm in Polynomialzeit lösen, bis auf einen Fehler (in der Zielfunktion) von ε. Im allgemeinen läßt sich so ein Fehlerterm nicht vermeiden, weil die optimale Lösung eines semidefiniten Programms irrational sein kann.
Es mag auch überraschen, dass man wissen muß, dass das Programm eine polynomiell beschränkte Lösung hat. Warum so etwas aber wichtig ist, sehen wir an folgendem Beispiel: Während wir für ein lineares Programm von vornherein wissen, dass bei gegebener Größe L des linearen Programms ein Lösungspunkt existiert, dessen Koordinaten betragsmäßig durch 2^L beschränkt sind, so gilt dies bei der Semidefiniten Programmierung nicht mehr. Betrachte folgendes Beispiel:

$$\textbf{min} \;\; x_n \;\; \textbf{so dass} \;\; x_1 = 2 \text{ und } x_i \geq x_{i-1}^2.$$

Die optimale Lösung dieses Programms ist natürlich $2^{2^{n-1}}$ und es handelt sich um ein semidefinites Programm, weil man die Bedingung $x_i \geq x_{i-1}^2$ in einem semidefiniten Programm formulieren kann (Übungsaufgabe). Allein zur Ausgabe der Lösung würden wir aber schon Laufzeit $\Omega(2^n)$ benötigen.
Wir haben erwähnt, dass die Semidefinite Programmierung eine Verallgemeinerung der Linearen Programmierung ist. Dies wollen wir beweisen:

4.26 Lemma. *Die Lineare Programmierung ist ein Spezialfall der Semidefiniten Programmierung.*

Beweis: Wir schreiben die Bedingungen des linearen Programms wie folgt auf. Die i-te Bedingung sei durch $IN_i \geq 0$ gegeben, wobei

$$IN_i := a_{i,1}x_1 + \cdots + a_{i,n}x_n + b_i$$

ist, für $1 \leq i \leq r$. Wir definieren die Matrix *SP* für das semidefinite Programm wie folgt:

$$\begin{pmatrix} IN_1 & 0 & \cdots & 0 \\ 0 & IN_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & IN_r \end{pmatrix}.$$

Da eine Diagonalmatrix genau dann PSD ist, wenn alle ihre Diagonalelemente nicht-negativ sind, erhalten wir, dass der zulässige Bereich des linearen Programms und des semidefiniten Programms gleich sind. □

Lineare Programmierung kann also nach diesem Beweis angesehen werden als der Spezialfall der Semidefiniten Programmierung, bei dem die beteiligte Matrix *SP* eine Diagonalmatrix ist.
Viele Eigenschaften, die für die Lineare Programmierung gelten, lassen sich auf die Semidefinite Programmierung übertragen. Zum Beispiel ist der zulässige Bereich eines semidefiniten Programms auch konvex, was durch eine einfache Rechnung überprüft werden kann.
Als eine Konsequenz aus der Konvexitätseigenschaft wollen wir hier festhalten, dass man z.B. ein semidefinites Programm nicht benutzen kann, um eine Bedingung wie zum Beispiel „$x \geq 1$ oder $x \leq -1$“ auszudrücken, da der zulässige Bereich für diese Menge nicht konvex ist.
Wir wollen an einem Beispiel zeigen, dass man mit Hilfe eines Semidefiniten Programms mehr Bedingungen ausdrücken kann als mit Hilfe eines Linearen Programms.
Wir betrachten zum Beispiel die symmetrische 2×2-Matrix

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

Diese Matrix ist genau dann PSD, wenn $a \geq 0, c \geq 0$, und $b^2 \leq ac$ ist. Zum Beispiel beschreibt damit die folgende Matrix

$$\begin{pmatrix} x_1 & 2 \\ 2 & x_2 \end{pmatrix}$$

als zulässigen Bereich denjenigen, wo für die Variablen x_1 und x_2 gilt, dass $x_1 \cdot x_2 \geq 4, x_1 \geq 0, x_2 \geq 0$ ist. Hier ist die Bedingung $x_1 \cdot x_2 \geq 4$ nicht linear.
Ein weiteres Beispiel: Wenn wir im semidefiniten Programm die Matrix

$$\begin{pmatrix} 1 & x & 0 \\ x & y & 0 \\ 0 & 0 & -y + \frac{x}{3} + \frac{1}{2} \end{pmatrix}$$

wählen, dann sind genau alle Punkte x, y mit $y \geq 0, y \geq x^2$ und $y \leq x/3 + 1/2$ zulässig. Diese Punkte liegen im folgenden Bild alle zwischen der Parabel und der Geraden. Wir überlassen es dem Leser und der Leserin, die optimale Lösung dieses semidefiniten Programms zu bestimmen, wenn man konkrete Zielfunktionen wählt.

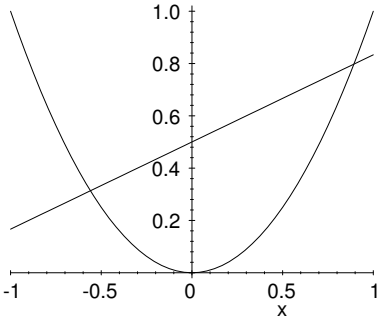


Abbildung 4.2: ***

4.3.5 MaxCut und Semidefinite Programmierung

Das Problem Maxcut ist wie folgt definiert:

Problem Maxcut Gegeben ein ungerichteter Graph $G = (V, E)$. Finde eine Zerlegung der Knotenmenge $V = V_1 \cup V_2$ (mit $V_1 \cap V_2 = \emptyset$), so dass die Anzahl der Kanten aus E, die zwischen V_1 und V_2 liegen, maximal wird.

Maxcut ist ein NP-hartes Problem, also ist man schon glücklich, wenn man Lösungen, d.h. Zerlegungen berechnen kann, die relativ gut sind, wenn auch nicht optimal.
Für ein $0 \leq c \leq 1$ heiße ein Approximationsalgorithmus A für Maxcut c-Approximationsalgorithmus, wenn die von A berechnete Lösung mindestens c mal soviele Kanten enthält wie die optimale Lösung. Bis ca. 1994 waren nur 1/2-Approximationsalgorithmen für Maxcut bekannt. So war es quasi eine Sensation, als es den beiden Forschern Goemans und Williamson gelang, einen Algorithmus zu entwerfen, der ein 0.878...-Approximationsalgorithmus für Maxcut ist. Insbesondere haben sie gezeigt, dass die Semidefinite Programmierung eine interessante Methode ist, um zu Approximationsalgorithmen zu gelangen, und entsprechend haben sie eine Lawine ausgelöst, in der viele weitere Probleme daraufhin untersucht wurden, ob sie mit Hilfe der Semidefiniten Programmierung besser gelöst werden können.

Wir beobachten zunächst, dass man die optimale Lösung eines MAXCUT-Problems als Lösung des folgenden mathematischen Programms erhalten kann:

$$\textbf{max} \sum_{\{i,j\}\in E} \frac{1-y_iy_j}{2} \quad \textbf{so dass} \quad y_i \in \{-1,1\} \text{ ist für alle } 1 \leq i \leq n.$$

Die Idee hinter dieser Formulierung ist, dass man $V_1 = \{i \mid y_i = 1\}$ und $V_2 = \{i \mid y_i = -1\}$ als die beiden Klassen der Zerlegung wählen kann. Ein Term $(1 - y_iy_j)/2$ trägt 1 zur Summe bei genau dann, wenn $y_i \neq y_j$ ist und 0 sonst.
Wir wollen dieses mathematische Programm „relaxieren“, d.h. den Raum der zulässigen Lösungen zunächst größer machen. Wir machen den Raum der zulässigen Lösungen dabei so groß, dass wir das neue Programm mit Hilfe der Semidefiniten Programmierung lösen können. Wir relaxieren das Programm dahingehend, dass wir von den Variablen y_i nicht verlangen, dass sie 1-dimensional sind, sondern, wir lassen zu, dass sie n-dimensional sind. Wir erhalten das folgende mathematische Programm:

$$\textbf{max} \sum_{\{i,j\}\in E} \frac{1-\underline{y_i}\,\underline{y_j}}{2} \quad \textbf{so dass} \quad ||\underline{y_i}|| = 1, \underline{y_i} \in \mathbf{R}^n \text{ ist für alle } 1 \leq i \leq n.$$

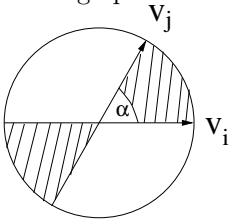
(Ausnahmsweise stellen wir in diesem Unterkapitel einmal die Vektoren durch unterstrichene Buchstaben dar, um sie deutlicher von Integervariablen zu unterscheiden.)
Das gerade gezeigte Programm ist noch kein semidefinites Programm, aber indem wir neue Variablen einführen, erhalten wir ein semidefinites Programm:

$$\textbf{max} \sum_{\{i,j\}\in E} \frac{1-y_{i,j}}{2} \quad \textbf{so dass} \quad Y := \begin{pmatrix} 1 & y_{1,2} & y_{1,3} & \dots & y_{1,n} \\ y_{1,2} & 1 & y_{2,3} & \dots & y_{2,n} \\ y_{1,3} & y_{2,3} & 1 & \dots & y_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{1,n} & y_{2,n} & y_{3,n} & \dots & 1 \end{pmatrix} \text{ PSD ist.}$$

Der Grund ist der folgende: Da man symmetrische Matrizen A, die PSD sind, als $A = B^T \cdot B$ zerlegen kann, definiert die Matrix einen zulässigen Bereich, in dem jede Variable $y_{i,j}$ geschrieben werden kann als das Produkt von Vektoren $\underline{v_i} \cdot \underline{v_j}$. Die Diagonale in der Matrix garantiert dabei, dass $||\underline{v_i}|| = 1$ für alle i ist.
Dies ist eine Relaxation des ursprünglichen Problems in einer Dimension, denn jeder zulässige Punkt mit Zielfunktionswert Z für das 1-dimensionale Problem kann als zulässiger Punkt mit Zielfunktionswert Z für das n-dimensionale Problem gesehen werden: Wir füllen die anderen Komponenten einfach mit Nullen.
Wir entwerfen nun folgenden randomisierten Algorithmus, um eine Zerlegung $V_1 \cup V_2$ zu bekommen.
Zunächst wenden wir einen Algorithmus an, um eine Lösung des obigen semidefiniten Programms zu finden, die optimal ist. (Bzw. bis auf einen kleinen additiven Term ε optimal ist). Diese Lösung besteht unter anderem aus der „optimal belegten“ Matrix Y. Wir fahren dann wie folgt fort:

- Berechne eine Cholesky-Zerlegung von Y und erhalte so Vektoren $\underline{v_i}$, $i = 1, \dots, n$, für die $y_{i,j} = \underline{v_i} \cdot \underline{v_j}$ gilt und $||v_i|| = 1$. Dies kann in Laufzeit $O(n^3)$ geschehen.
- Wähle zufällig eine Hyperebene H aus. Dies kann zum Beispiel dadurch geschehen, dass man einen Zufallsvektor $\underline{r}$ als Normale der Hyperebene auswürfelt. Zu diesem Zweck benutzt man eine rotationssymmetrische Verteilung.
- Wir wählen V_1 als die Menge all derjenigen Knoten aus V, deren zugehöriger Vektor auf einer Seite der Hyperebene H liegt. Also: $v_i \in V_1 \iff \underline{r} \cdot \underline{v_i} \leq 0$. Außerdem wählen wir $V_2 := V \setminus V_1$.

Der Vorgang, aus den Vektoren $\underline{v_i}$ mittels der Wahl einer Hyperebene Elemente aus $\{-1, 1\}$ zu machen, wird auch „Rundungsprozedur“ genannt.
Der gerade beschriebene Algorithmus ist ein randomisierter Algorithmus, er hängt also von Zufallsentscheidungen ab, nämlich von der Wahl der Hyperebene. Die berechnete Zerlegung ist eine Zufallsvariable. Zufallsentscheidungen können auch schlecht ausgehen, das heißt, wir bekommen nicht unbedingt jedes Mal eine gute Zerlegung. Was wir aber zeigen können, ist folgendes: Die erwartete Kantenzahl der zufällig berechneten Zerlegung ist mindestens 87% der Kantenzahl in einer optimalen Zerlegung. Dieser Aufgabe wollen wir uns nun widmen.
Wegen der Linearität des Erwartungswertes brauchen wir nur für zwei gegebene Vektoren $\underline{v_i}$ und $\underline{v_j}$ die Wahrscheinlichkeit auszurechnen, dass sie auf verschiedenen Seiten der gewählten Hyperebene liegen. Da das Produkt $\underline{v_i} \cdot \underline{v_j}$ rotationsinvariant ist, können wir die Ebene H^* betrachten, die von den beiden Vektoren aufgespannt wird:



(Der Winkel $\alpha = \arccos(\underline{v_i} \cdot \underline{v_j})$ zwischen den Vektoren $\underline{v_i}$ und $\underline{v_j}$ ist rotationsinvariant.)
Als erstes beobachten wir folgendes: die Wahrscheinlichkeit, dass die gewählte Hyperebene H gleich H^* ist, ist gleich Null.
In den anderen Fällen schneiden sich H und H^* in einer Geraden. Wenn die beiden Vektoren auf verschiedenen Seiten von H liegen sollen, dann müssen sie auch auf verschiedenen Seiten der Geraden liegen. Also muß die Gerade „zwischen“ $\underline{v_i}$ und $\underline{v_j}$ liegen, also in den Bereich fallen, der im Bild schraffiert ist. Dies passiert jedoch genau mit Wahrscheinlichkeit

$$\frac{2\alpha}{2\pi} = \frac{\alpha}{\pi} = \frac{\arccos(\underline{v_i} \cdot \underline{v_j})}{\pi}.$$

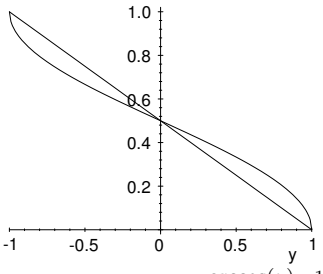
Also ist die erwartete Zahl an Kanten, die zwischen V_1 und V_2 liegen („Schnittkanten“ genannt) in der zufälligen Zerlegung $V = V_1 \cup V_2$ wie folgt zu berechnen:

$$\mathbf{E}\,[\text{Anzahl Schnittkanten}] \quad = \quad \sum_{\{i,j\} \in E} \frac{\arccos(\underline{v_i} \cdot \underline{v_j})}{\pi}. \tag{4.1}$$

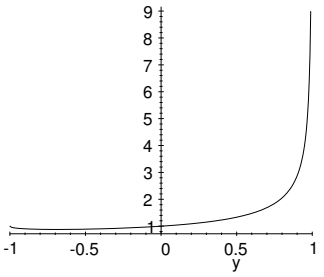
Wenn wir eine konkrete semidefinite Relaxation des Problems MAXCUT berechnen, erhalten wir natürlich eine obere Schranke für die optimale MAXCUT-Lösung. Die Güte der Lösung hängt vom Ausgang der Rundungsprozedur ab, wir berechnen die erwartete Güte. Im konkreten Fall kann man die erhaltene Lösung mit der oberen Schranke vergleichen und so direkt die Güte der erhaltenen Lösung abschätzen. Entsprechend kann man bei Bedarf weitere Zufallsversuche vornehmen.
Als Erfahrungswert sollte man auch im Hinterkopf behalten, dass Experimente zeigen, dass in den „meisten“ Fällen die berechnete Lösung sogar eine Güte hat, die viel besser ist als die von uns bewiesenen $0.878\ldots$.
Wir wollen den Erwartungswert in Ausdruck (4.1) mit dem Wert des relaxierten Programms vergleichen. Eine Möglichkeit, dies zu tun, besteht darin, jeden einzelnen Term $\arccos(\underline{v_i} \cdot \underline{v_j})/\pi$ mit $(1 - \underline{v_i} \cdot \underline{v_j})/2$ zu vergleichen. Es ist eine relativ einfache Analysisaufgabe, zu zeigen, dass im Bereich $-1 \leq y \leq 1$

$$\frac{\arccos(y)}{\pi} \geq 0.87856 \cdot \frac{1-y}{2} \quad \text{gilt.}$$

Im folgenden wollen wir die beteiligten Funktionen bildlich darstellen: Wir sehen im folgenden Bild die beiden Funktionen $\arccos(y)/\pi$ und $(1-y)/2$.



Die folgenden beiden Bilder zeigen den Quotienten $\frac{\arccos(y)}{\pi} / \frac{1-y}{2}$ im Bereich $-1 \leq y \leq 1$ (links) und $-1 \leq y \leq 0$ (rechts).



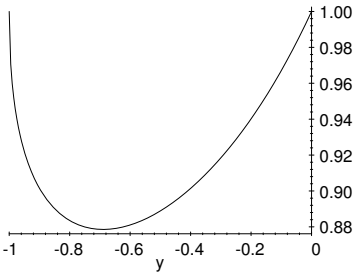


Abbildung 4.3: ***

Es bezeichne $Relax_{opt}$ den optimalen Wert der Zielfunktion des Semidefiniten Programms. Die gerade gemachte Analyse zeigt, dass der Erwartungswert der Anzahl der gefundenen Schnittkanten mindestens $0.878 \cdot Relax_{opt}$ ist. Da $Relax_{opt}$ eine obere Schranke für die maximale Anzahl an Schnittkanten ist, gilt also, dass wir eine zufällige Zerlegung $V_1 \cup V_2$ erhalten, die im Erwartungswert mindestens 0.878 mal so viele Schnittkanten hat wie die optimale Zerlegung.
Zum Schluß wollen wir betonen, dass wir gerade zwar einen randomisierten Algorithmus für die Rundungsprozedur beschrieben haben, dass aber auch deterministische Algorithmen mit der gleichen Güte bekannt sind. Diese sind jedoch wesentlich schwieriger zu beschreiben als die obige Rundungsprozedur.

4.4 Die probabilistische Methode

Die probabilistische Methode wird benutzt, um die Existenz von gewissen Objekten nachzuweisen (mit positiver Wahrscheinlichkeit). Als erstes betrachten wir sogenannte „Expandergraphen“ (speziell: Konzentratoren), die u.a. im Gebiet der Kommunikationsnetzwerke und bei der Derandomisierung von Algorithmen Anwendung finden. Intuitiv ist ein Expandergraph ein bipartiter Graph, bei dem man die Gewissheit hat, dass gewisse Knotenmengen „links“ große Nachbarmengen „rechts“ haben. Formal heißt das:

4.27 Definition. Ein bipartiter Graph $G = (L \cup R, E)$ heißt (n, d, α, c)-*OR-Konzentrator*, wenn die folgenden drei Bedingungen alle erfüllt sind:

1. $|L| = |R| = n$.
2. Jeder Knoten aus L hat Grad höchstens d.
3. Jede Teilmenge $S \subseteq L$ mit $|S| \leq \alpha \cdot n$ hat mindestens $c \cdot |S|$ viele Nachbarn in R.

Wenn ein Graph G ein (n, d, α, c)-OR-Konzentrator ist, dann gilt notwendigerweise, dass

$$c \leq n/\lfloor \alpha n \rfloor \approx 1/\alpha$$

ist, denn für eine Teilmenge S mit $|S| = \lfloor \alpha n \rfloor$ muß R mindestens $c \cdot \lfloor \alpha n \rfloor$ viele Nachbarn haben, R enthält aber nur n Knoten. Auf ähnliche Art folgt $c \leq d$. Diese beiden Werte kann c maximal annehmen, aber die meisten Konstruktionen sind schlechter. Typischerweise interessieren übrigens die Konzentratoren mit $c > 1$.

4.28 Behauptung. Für alle $n \geq 1$ gibt es einen $(n, 18, \frac{1}{3}, 2)$-OR-Konzentrator.

Beweis: Wir führen den Beweis probabilistisch. Die auftretenden Parameter fixieren wir erst am Ende des Beweises. L enthält n Knoten, R enthält n Knoten. Wir würfeln einen zufälligen bipartiten Graphen auf $L \cup R$ aus, indem wir die Kanten zufällig auswürfeln. Für jeden Knoten $v \in L$ wählen wir d mal (mit Ersetzung) einen Knoten w auf der rechten Seite aus und fügen die Kante $\{v, w\}$ hinzu (wenn sie nicht schon da ist). Damit ist gewährleistet, dass jeder Knoten aus L Grad höchstens d hat. Wir zeigen nun, dass folgendes gilt: *Mit Wahrscheinlichkeit mindestens* 0.8 *ist der ausgewürfelte Graph ein* $(n, 18, \frac{1}{3}, 2)$*-OR-Konzentrator.*
Sei nun $\alpha := 1/3, d := 18$ und $c := 2$ und nehmen wir an, dass G kein (n, d, α, c)-OR-Konzentrator ist.
Dann existiert eine Teilmenge $S \subseteq L$ mit $|S| \leq \alpha \cdot n$ und eine Teilmenge $T \subseteq R$ mit $|T| = c \cdot |S|$, so dass alle Nachbarn der Menge S in der Menge T enthalten sind. (Hier benötigen wir, dass $c|S|$ eine natürliche Zahl ist.) Sei S fest gewählt, $s := |S| \leq \alpha n$.
Die Wahrscheinlichkeit p_S, dass die Menge S nicht genügend Nachbarn hat, ist also beschränkt durch

$$p_S \leq \binom{n}{cs} \left(\frac{cs}{n}\right)^{ds},$$

wobei $\binom{n}{cs}$ die Anzahl Möglichkeiten angibt, T zu wählen. Der Term $\left(\frac{cs}{n}\right)^{ds}$ ergibt sich wie folgt: Für jeden Knoten aus S werden d Nachbarn gewählt, dies sind also insgesamt ds Wahlen. Jedesmal muss ein Knoten aus T gewählt werden, die Wahrscheinlichkeit, einen Knoten aus T zu wählen, ist cs/n.
Sei p_s nun die Wahrscheinlichkeit, dass es ein S mit $|S| = s$ gibt, das nicht genügend Nachbarn hat (also weniger als cs). Dann erhalten wir (die Erklärung für die jeweiligen Ungleichungen im Anschluss):

$$\begin{aligned}
p_s &\leq \binom{n}{s}\binom{n}{cs}\left(\frac{cs}{n}\right)^{ds} \\
&\leq \left(\frac{ne}{s}\right)^{s}\left(\frac{ne}{cs}\right)^{cs}\left(\frac{cs}{n}\right)^{ds} \\
&= (n^{1+c-d}e^{c+1}c^{d-c}s^{d-1-c})^{s} \\
&= \left[\left(c\frac{s}{n}\right)^{d-c-1}e^{c+1}c\right]^{s} \\
&\leq \left[\left(\frac{c}{3}\right)^{d-c-1}ce^{c+1}\right]^{s}
\end{aligned}$$

Bei der ersten Ungleichung haben wir die Abschätzung $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^{k}$ benutzt. Die letzte Ungleichung gilt, da $s \leq \alpha n$ und $\alpha = \frac{1}{3}$ ist. Damit haben wir eine obere Schranke für die Wahrscheinlichkeit, dass eine Menge S mit $|S| = s$ existiert, so dass die OR-Konzentratoreigenschaft nicht erfüllt ist. Da nach Voraussetzung $c = 2$ und $d = 18$ ist, ergibt sich $\left[\left(\frac{2}{3}\right)^{15}2e^{3}\right]^{s} = (c^*)^s$ mit $c^* \approx 0,091737$. Bisher haben wir s fest gewählt, jetzt müssen wir noch über alle s summieren:

Die Wahrscheinlichkeit, dass der gewürfelte Graph kein OR-Konzentrator mit den gewünschten Parametern ist, beträgt höchstens

$$\sum_{s=1}^{\lfloor \alpha n \rfloor} p_s \leq \sum_{s=1}^{\lfloor \alpha n \rfloor} (c^*)^s \leq \sum_{s=1}^{\infty} (c^*)^s = \frac{c^*}{1-c^*} \leq 0.11.$$

Die Wahrscheinlichkeit, dass der gewürfelte Graph ein OR-Konzentrator ist, beträgt damit mindestens 0.89. □

Übrigens kann man die obige Rechnung auch mit $d = 14$ durchführen und wird sehen, dass die Wahrscheinlichkeit, einen OR-Konzentrator mit entsprechendem Parameter zu bekommen, auch immer noch größer als 0 ist.
Anmerkung:

- Es ist kein effizienter deterministischer Algorithmus bekannt, um solche OR-Konzentratoren zu konstruieren.
- Es ist weiterhin kein effizienter Algorithmus bekannt, um zu verifizieren, dass ein gegebener Graph ein OR-Konzentrator ist.

Diese Nachteile führen dazu, dass man in den „Anwendungen“ andere Expanderbegriffe wählt. Der folgende Satz enthält einen alternativen Expanderbegriff.

4.29 Satz. *Es gibt für alle $n \geq 5$ einen bipartiten Graph $G = (L \cup R, E), L \cap R = \emptyset$ mit $|L| = n, |R| = 2^{\lfloor \log^2 n \rfloor}$, der die folgenden beiden Bedingungen erfüllt.*

i) Jeder Knoten $v \in R$ hat höchstens Grad $\lfloor \log^2 n \rfloor$.

ii) Jede Teilmenge $S \subseteq L$ mit $|S| \geq \frac{n}{2}$ hat mindestens $|R| - n$ viele Nachbarn.

Beweis: Als Abkürzung definieren wir $d := \lfloor \log^2 n \rfloor$. Würfle für jeden Knoten $v \in R$ dessen (höchstens d viele) Nachbarn aus L aus (mit Ersetzung, gemäß der Gleichverteilung).

- zu i) trivialerweise erfüllt.
- zu ii) Betrachte ein festes $S \subseteq L$ mit $|S| \geq \frac{n}{2}$. Die Zahl N_S sei die Anzahl der Knoten in R, die keinen einzigen Nachbarn in der Menge S haben. Um N_S zu analysieren, betrachten wir einen beliebigen Knoten $v \in R$. Die Wahrscheinlichkeit, dass v *keinen* Knoten aus S als Nachbar hat, ist $(1-|S|/|L|)^d \leq \left(\frac{1}{2}\right)^d$. Für N_S gilt somit $\mathbf{E}[N_S] \leq |R| \left(\frac{1}{2}\right)^d = 2^d \left(\frac{1}{2}\right)^d = 1$. Die Wahrscheinlichkeit, dass N_S grösser als n ist, lässt sich nun mit der Chernoffschranke (da $n \geq 5$ ist) abschätzen als $\mathbf{Prob}(N_S > n) \leq 2^{-n}$. Summieren wir nun über alle S mit $|S| \geq \frac{n}{2}$, (davon gibt es weniger als 2^n viele), dann erhalten wir:
  Die Wahrscheinlichkeit, dass der gewürfelte Graph nicht die gewünschte Eigenschaft hat, ist kleiner als $2^n 2^{-n} = 1$. Also existiert ein solcher Graph.

□

Expandergraphen und Wahrscheinlichkeitsamplifikation von RP-Algorithmen
Gegeben sei ein RP-Algorithmus A, der folgendermaßen arbeitet. A würfelt $r \in \{0, \ldots, n-1\}$ und berechnet deterministisch $A^*(x, r) \in \{0, 1\}$. Da A ein RP-Algorithmus ist, gelten folgende Bedingungen:

$$\begin{aligned} x \in L &\Rightarrow \mathbf{Prob}(A^*(x,r) = 1) \geq \frac{1}{2} \\ x \notin L &\Rightarrow \mathbf{Prob}(A^*(x,r) = 1) = 0 \end{aligned}$$

Bei T-facher Wiederholung erhält man eine durch $(\frac{1}{2})^T$ beschränkte Fehlerwahrscheinlichkeit. Mit $T \geq \log n$ ist die Fehlerwahrscheinlichkeit dann höchstens $1/n$, allerdings braucht man für $\log n$ Wiederholungen $\log^2 n$ viele Zufallsbits. Wir zeigen nun, wie man mit $\log^2 n$ Zufallsbits einen wesentlich kleineren Fehler erzielen kann.
Gegeben sei nun ein Expandergraph G gemäß der Definition aus dem obigen Theorem. Die Knoten in L seien o.B.d.A. mit $0, \ldots, n-1$ bezeichnet.

Annahme: Bei Eingabe $v \in R$ können dessen Nachbarn auf der linken Seite in Polynomialzeit berechnet werden.

Algorithmus NEU: Wir gehen davon aus, dass $L = \{0, \ldots, n-1\}$ ist und benutzen $\lfloor \log^2 n \rfloor$ Zufallsbits zur Adressierung eines Knotens $v \in R$. Wir berechnen effizient die Nachbarmenge S dieses Knotens.
Berechne $A^*(x, s)$ für alle $s \in S$. Falls mindestens einmal eine 1 berechnet wird, gib 1 aus, 0 sonst.

r heißt Zeuge für $x \in L$ genau dann, wenn $A^*(x, r) = 1$ ist. Wenn $x \in L$ ist, existieren mindestens $n/2$ Zeugen für x. Wenn $x \notin L$ ist, existieren überhaupt keine Zeugen für x. Wir zeigen nun:

a) $x \in L \Rightarrow \mathbf{Prob}\,(\mathrm{NEU}(x) = 1) \geq 1 - 1/n^{\log n - 2}$

b) $x \notin L \Rightarrow \mathbf{Prob}\,(\mathrm{NEU}(x) = 1) = 0$

- b) ist trivial, weil $x \notin L \Rightarrow A^*(x, r) = 0$ für alle r.
- zu a): Wähle Z als Menge der Zeugen. Dann gilt $|Z| \geq \frac{n}{2}$, also hat Z auf der rechten Seite mindestens $2^{\lfloor \log^2 n \rfloor} - n$ Nachbarn. Daher ist die Wahrscheinlichkeit, dass NEU eine 0 ausgibt, höchstens

$$\frac{n}{2^{\lfloor \log^2 n \rfloor}} \leq \frac{n}{2^{\log^2 n - 1}} = \frac{n}{n^{\log n - 1/\log n}} \leq \frac{1}{n^{\log n - 2}}.$$

Ein kritische(r) Zeitgenosse/in könnte anmerken, dass der Expandergraph „implizit“ Zufallsbits enthält. Allerdings kann *ein* Expandergraph für beliebige RP-Algorithmen, also immer wieder, verwendet werden.
Leider können auch diese Expandergraphen weder effizient berechnet werden, noch können die geforderten Eigenschaften für gegebene Graphen effizient verifiziert werden.

4.4.1 Reduktion der benötigten Zufallsbits beim Oblivious Routing

In Abschnitt 4.1 haben wir folgendes festgestellt:

1.) Jeder deterministische OPRA benötigt im worst case auf dem Hyperwürfel $\Omega(\frac{2^n}{n})$ Schritte.

2.) Es existiert ein randomisierter OPRA, der auf jeder Permutation eine erwartete Laufzeit von höchstens $10n$ hat.

Genau genommen hatten wir die zweite Aussage nur implizit gezeigt, wir machen ihren Beweis in den folgenden Zeilen expliziter:
Wir hatten in Satz 4.7 gezeigt, dass die Wahrscheinlichkeit, dass jedes Paket nach höchstens $8n$ Schritten sein Ziel erreicht, mindestens $1 - \frac{1}{N}$ beträgt. Wie hängt das mit 2.) zusammen? Der Erwartungswert für die Laufzeit ist höchstens

$$\left(1 - \frac{1}{N}\right) 8n + \frac{1}{N}(2Nn) \leq 10n.$$

Der Grund dafür ist der folgende: Mit Wahrscheinlichkeit mindestens $1 - (1/N)$ haben wir höchstens $8n$ Schritte. Mit Wahrscheinlichkeit höchstens $1/N$ können wir aber mehr als $8n$ Schritte machen. Wieviele Schritte maximal? Die obige Gleichung behauptet $2Nn$ viele Schritte maximal. Dies zeigen wir nun:
Jedes Paket hat einen Weg der Länge maximal $2n$ zu laufen. (Grund: Zum Zwischenziel höchstens n Kanten entlanglaufen und von dort zum Endziel auch höchstens n Kanten entlanglaufen.) Da wir N Pakete vorliegen haben und pro Zeiteinheit mindestens ein Paket eine Kante entlangläuft, dauert das Routen maximal $N \cdot 2n$ Schritte.

Unser randomisierter OPRA verwendet $N \log N = Nn$ viele Zufallsbits, da für jedes der N Pakete ein Zwischenziel gewählt wird.
Ein deterministischer OPRA kann auch aufgefasst werden als randomisierter OPRA, der 0 Zufallsbits benutzt. Damit haben wir einen Tradeoff:
Randomisierte OPRAs, die

- keine Zufallsbits verwenden, haben worst-case-Laufzeit $\Omega(\sqrt{2^n/n})$.
- $N \cdot n$ Zufallsbits verwenden dürfen, können mit Laufzeit $O(n)$ auskommen.

Mit wievielen Zufallsbits kann man lineare erwartete Schrittzahl erreichen? Dieser Frage widmen wir uns im folgenden.
Es gibt *endlich* viele deterministische OPRAs. Input für ein OPRA ist eine Permutation $\pi \in \Sigma_N$ auf N Orten. Ein OPRA legt zu jeder Permutation und für alle Startorte die Route zum Zielort fest, kann also als Abbildung

$$f : \Sigma_N \times \{1, \ldots, N\} \to \{1, \ldots, N\}^{\leq N}$$

aufgefasst werden. Da die beteiligten Mengen alle endlich sind, gibt es nur endlich viele solche Abbildungen, also endlich viele deterministische OPRAs. Damit ist eine wichtige Voraussetzung erfüllt, um Yaos Minimaxprinzip verwenden zu dürfen.
Für randomisierte OPRAs heißt das: ein randomisierter OPRA ist eine Wahrscheinlichkeitsverteilung $p_1, \ldots, p_R$ über deterministische OPRAs $A_1, \ldots, A_R$.

4.30 Beispiel. Im vorgestellten randomisierten OPRA würfelt man die Funktion $\sigma : \{1, \ldots, N\} \to \{1, \ldots, N\}$ aus und benutzt dann einen deterministischen OPRA A_σ. Der randomisierte OPRA A würfelt also einen von N^N vielen deterministischen OPRAs aus.

4.31 Satz. *Ein randomisierter OPRA auf dem Hyperwürfel, der auf jeder Eingabe höchstens k Zufallsbits verwendet, hat erwartete Schrittzahl mindestens*

$$\Omega\left(2^{-k} \cdot \sqrt{\frac{N}{n}}\right).$$

Beweis: Wenn höchstens k Zufallsbits gewürfelt werden, wählt der randomisierte OPRA einen von höchstens 2^k vielen deterministischen Algorithmen aus. Also gibt es einen *deterministischen* OPRA A^*, der mit Wahrscheinlichkeit mindestens 2^{-k} aufgerufen wird. Sei x eine worst-case-Eingabe für A^*. Dann hat A^* auf x Laufzeit mindestens $\sqrt{2^n/n}$. Der randomisierte Algorithmus hat erwartete Laufzeit mindestens $\Omega(2^{-k} \cdot \sqrt{2^n/n})$. □

Wie groß muss k sein, damit $2^{-k} \cdot \sqrt{2^n/n}$ linear ist? $k = \Omega(n)$ ist nötig. Damit klafft eine Lücke zwischen der Anzahl der Zufallsbits, die unser randomisierter Algorithmus oben benutzt hat und der unteren Schranke für die Mindestzahl an Zufallsbits. Wir zeigen nun, dass die *untere* Schranke scharf ist.

4.32 Satz. *Für alle n gibt es einen randomisierten OPRA, der auf dem Hyperwürfel mit $3n$ Zufallsbits auskommt und erwartete Schrittzahl höchstens $12n$ hat.*

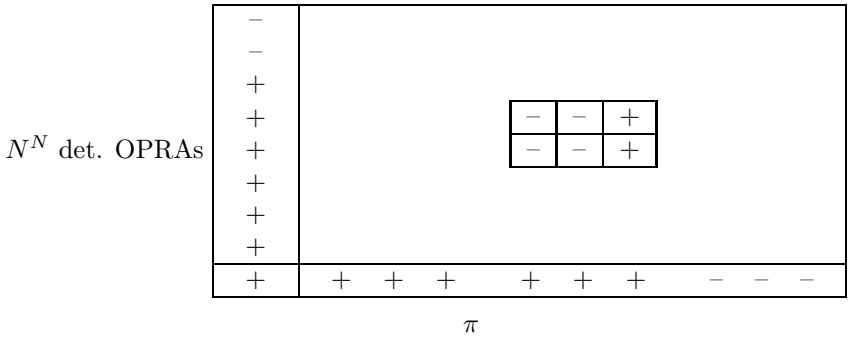
Unser Vorgehen wird wie folgt sein: Ein Algorithmusdesigner wählt $A_1, \ldots, A_{N^3}$ („gute Auswahl“) als deterministische OPRAs fest. Bei Eingabe x wählt er einen der Algorithmen gemäß der Gleichverteilung aus.

$$\log N^3 = 3 \cdot \log N = 3n.$$

Uns bleibt zu zeigen, dass es eine gute Auswahl von N^3 vielen OPRAs gibt.

4.33 Definition. Ein deterministischer OPRA heißt „schlecht“ für eine Permutation π, falls das Routen von π mehr als $8n$ Schritte braucht.

Laut Satz 4.7 ist für ein gewähltes π nur ein Anteil von $1/N$ der OPRAs schlecht für π. Wir machen uns folgendes Bild, die Zeilen stehen für die deterministischen OPRAs, die Spalten für die Permutationen und ein „+“ bzw. „−“ gibt an, ob das OPRA gut oder schlecht für π ist.



Pro Spalte ist höchstens ein Anteil von $1/N$ der OPRAs schlecht.
Wir zeigen nun, dass man im Prinzip mit einer viel kleineren Matrix auskommen kann, dass man nämlich mit nur N^3 deterministischen Algorithmen auskommt. Wir benutzen die probabilistische Methode - zeigen also nur, dass es eine solche Auswahl gibt.
Zu diesem Zweck wählen wir aus den höchstens N^N vielen deterministischen OPRAs gemäß der Gleichverteilung mit Zurücklegen N^3 deterministische OPRAs aus: $A_1, \ldots, A_{N^3}$.
Für die Analyse sei nun eine Permutation π fix gewählt.
Wieviele der OPRAs $A_1, \ldots, A_{N^3}$ sind schlecht für π? Sei X_i die Indikatorvariable, die 1 ist, falls A_i schlecht ist für π. Dann ist $X = \sum_{i=1}^{N^3} X_i$ die Anzahl der schlechten OPRAs für π. Es gilt

$$\mathbf{E}[X] \leq N^2,$$

weil $\mathbf{E}[X_i] \leq \frac{1}{N}$ ist. Wie groß ist die Wahrscheinlichkeit, dass $X > 2N^2$ ist? Es ist

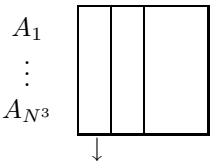
$$\mathbf{Prob}\left(X > 2N^2\right) \leq \mathbf{Prob}\left(X \geq 2N^2\right) = \mathbf{Prob}\left(X \geq (1+1) \cdot m\right),$$

für $m = N^2 \geq \mathbf{E}[X]$. Wir verwenden die Chernoffschranke in der „a0-Variante“ mit $m = N^2$ und $\delta = 1$ und erhalten $\mathbf{Prob}\left(X \geq 2N^2\right) \leq (e/4)^{N^2}$.
Eine Permutation π heiße schlecht für $A_1, \ldots, A_{N^3}$ genau dann, wenn mehr als $2N^2$ der A_i schlecht sind für π. Wir wollen die Versagenswahrscheinlichkeit $V := \mathbf{Prob}\left(\exists\, \pi, \text{ die schlecht ist für } A_1, \ldots, A_{N^3}\right)$ ausrechnen. Da es $N!$ Permutationen gibt, können wir wieder wegen der Eigenschaft $\mathbf{Prob}(A \cup B) \leq \mathbf{Prob}(A) + \mathbf{Prob}(B)$ abschätzen:

$$V \leq N! \cdot (e/4)^{N^2} < 1.$$

Die letzte Ungleichung ergibt sich wie folgt:

$N! \leq N^N$, also $\ln N! \leq N \ln N$. Es reicht also aus, zu zeigen, dass $N \ln N + (N^2) \cdot \ln(e/4) < 0$ ist. Dass dies asymptotisch gilt, ist auf den ersten Blick klar, dass es auch für alle $N \geq 1$ gilt, kann man durch eine einfache zweizeilige Kurvendiskussion nachweisen, welche wir natürlich weglassen.
Es gibt also eine Algorithmenauswahl $A_1, \ldots, A_{N^3}$, die für *keine* der Permutationen schlecht ist.



pro Spalte $\leq 2N^2$ viele „–“-Zeichen.

$$\frac{2N^2}{N^3} = \frac{2}{N}.$$

Ein Algorithmusdesigner fixiert diese Auswahl. Zur erwarteten Schrittzahl von höchstens $12n$ kommen wir wie gehabt: Mit Wahrscheinlichkeit $(1 - 2/N)$ höchstens $8n$ Schritte, mit Wahrscheinlichkeit $2/N$ höchstens $2nN$ Schritte.

Algorithmus: Mit $\log N^3 = 3n$ vielen Bits wähle bei vorliegender Eingabe π einen dieser N^3 OPRAs aus. Der so gewonnene randomisierte OPRA hat erwartete Schrittzahl höchstens $12n$.

Nachteil: Die Konstruktion ist nicht-uniform. Wenn N gegeben ist, ist die Frage, welche der N^3 Algorithmen ausgewählt werden sollen.

4.5 Derandomisierung

Es sei ein randomisierter Algorithmus gegeben. Derandomisierung bezeichnet den Vorgang, ihn auf *mechanische* Weise in einen deterministischen Algorithmus umzuwandeln. Eine spezielle Methode: die Potenzialmethode, diese umfasst zum Beispiel:

- die Methode des pessimistischen Schätzers
- die Methode der bedingten Wahrscheinlichkeiten

Wir hatten früher in der Vorlesung folgendes Beispiel:
Berechne eine unabhängige Menge in einem ungerichteten Graphen $G = (V, E)$, der Durchschnittsgrad $d_{avg} \geq 1$ hat. Wir hatten gezeigt: Es gibt eine unabhängige Menge I mit $|I| \geq \frac{n}{2 \cdot d_{\text{avg}}}$. Es seien gegeben die Wahrscheinlichkeiten $p_1, \ldots, p_n$, wähle Knoten i mit Wahrscheinlichkeit p_i.

$$\mathbf{E}[|I|] \geq p_1 + \cdots + p_n - \sum_{e=\{i,j\} \in E} p_i \cdot p_j$$

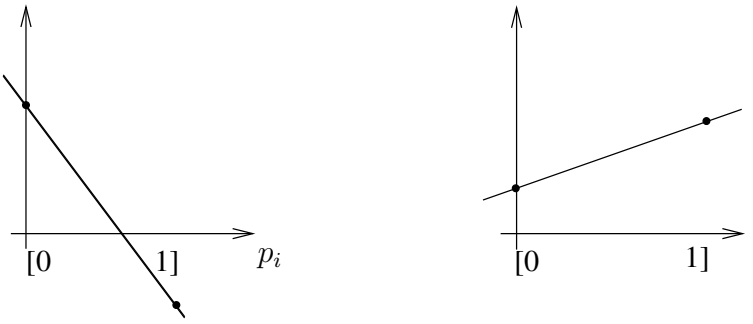
$$p_1 = p_2 = \ldots = p_n = \frac{1}{d_{\text{avg}}}$$

Wähle als „Potenzialfunktion“

$$V(p_1, \ldots, p_n) := p_1 + \cdots + p_n - \sum_{e=\{i,j\} \in E} p_i \cdot p_j.$$

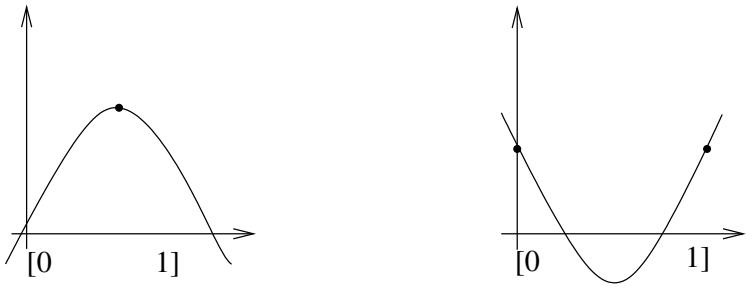
- V ist linear in jedem p_i.
- V ist *effizient* auszurechnen!

Es gilt $V(1, p_2, \ldots, p_n) \geq V(p_1, \ldots, p_n)$ oder $V(0, p_2, \ldots, p_n) \geq V(p_1, \ldots, p_n)$.
Deterministischer Algorithmus: Seien $p_1 = p_1^*, \ldots, p_n = p_n^*$ gewählt (z.B. $p_i = \frac{1}{d_{\text{avg}}}$).



For $i = 1$ to n do
    begin
        if $V(p_1, \ldots, p_{i-1}, 1, p_{i+1}, \ldots, p_n) \geq V(p_1, \ldots, p_{i-1}, 0, p_{i+1}, \ldots, p_n)$
            then $p_i = 1$ else $p_i = 0$;
    end;
$I := \{v_i \mid p_i = 1\}$.
Für jede Kante auf I entferne einen inzidenten Knoten aus I.

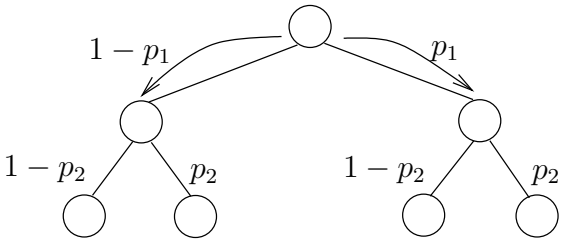
Es gilt am Schluss $V(p_1, \ldots, p_n) \geq V(p_1^*, \ldots, p_n^*)$. I enthält mindestens $V(p_1^*, \ldots, p_n^*) \geq \frac{n}{2 \cdot d_{\text{avg}}}$ viele Knoten. Die Methode der Potenzialfunktionen funktioniert prinzipiell auch für Potenzialfunktionen, die die Eigenschaft haben, dass für jedes p_i das Maximum der Potenzialfunktion entweder für $p_i = 0$ oder für $p_i = 1$ angenommen wird. Damit darf die Potenzialfunktion in einem Parameter p_i auch z.B. eine nach oben offene Parabel sein, nicht aber eine nach unten offene Parabel.



Methode der bedingten Wahrscheinlichkeiten Diese ist ein Spezialfall der Methode der Potenzialfunktionen. Als Potenzialfunktion wählt man nämlich (bedingte) Wahrscheinlichkeiten. Angenommen, wir haben einen randomisierten Algorithmus, dessen Ablauf durch Parameter $p_1, \ldots, p_n$ beschrieben ist. Den Ablauf des Algorithmus stellen wir uns durch einen binären Baum beschrieben vor. Gestartet wird in der Wurzel. Wenn wir auf Ebene i sind (Wurzel auf Ebene 1), dann verzweigt der Algorithmus mit Wahrscheinlichkeit p_i nach rechts, mit der Wahrscheinlichkeit $1 - p_i$ nach links. An den Blättern hat der Algorithmus entweder „Erfolg“ oder keinen Erfolg.
Mit $P(p_1, \ldots, p_n)$ können wir nun die Wahrscheinlichkeit beschreiben, dass der Algorithmus an einem Blatt ankommt, an dem er Erfolg hat. Aus dem Baum sehen wir:

$$P(p_1, \ldots, p_n) = p_1 \cdot P(1, p_2, \ldots, p_n) + (1 - p_1) \cdot P(0, p_2, \ldots, p_n).$$

Komplettes Hinschreiben der Formel für P würde übrigens 2^n Terme erfordern, man sieht aber auch, dass sie in jeder Variable linear ist.



Wenn man Parameter $p_1, \ldots, p_n$ kennt, für die $P(p_1, \ldots, p_n) > 0$ ist, dann könnte man den randomisierten Algorithmus durchführen und hätte mit von Null verschiedener Wahrscheinlichkeit Erfolg.
Angenommen, wir könnten für alle $p_1, \ldots, p_n$ auch noch die Wahrscheinlichkeit $P(p_1, \ldots, p_n)$ effizient berechnen, dann könnten wir auch auf effiziente *deterministische* Art und Weise ein erfolgreiches Blatt finden:
for $i = 1$ **to** n **do**
if $P(p_1, \ldots, p_{i-1}, 1, p_{i+1}, \ldots, p_n) \geq P(p_1, \ldots, p_{i-1}, 0, p_{i+1}, \ldots, p_n)$
then setze $p_i = 1$
else setze $p_i = 0$.

Am Schluss ist der P-Wert mindestens so groß wie zu Beginn. Wenn dieser zu Beginn $P(p_1, \ldots, p_n)$ größer als 0 ist, dann ist am Schluss mit den veränderten p_i der Wert $P(p_1, \ldots, p_n) > 0$. Am Schluss *befinden* wir uns also an einem *guten* Blatt.
Wieso der Name „Methode der bedingten Wahrscheinlichkeiten“? Dieser rührt im Prinzip daher, dass man (z.B.) die Wahrscheinlichkeit $P(1, p_2, \ldots, p_n)$ auch ansehen kann als „Erfolgswahrscheinlichkeit, wenn man am rechten Kind der Wurzel startet“, was wiederum als bedingte Wahrscheinlichkeit gelesen werden kann.

4.5.1 Methode des pessimistischen Schätzers

Falls die Versagenswahrscheinlichkeit $P(p_1, \ldots, p_n)$ *nicht* exakt berechenbar oder *nicht* linear in allen p_i ist, reicht es manchmal, eine obere Schranke für die Versagenswahrscheinlichkeit $P(p_1, \ldots, p_n) \leq Q(p_1, \ldots, p_n)$ zu berechnen, wobei Q linear in allen p_i und effizient berechenbar ist. Nun lassen sich die Verfahren wie bisher auf Q anwenden.
Nun kommen wir zu Beispielen für die genannten Derandomisierungsmethoden.
Beispiel 1: (eigentlich unsinnig, weil greedy-Methode genauso gut) MAXCUT:
Wir würfeln (für alle i) Knoten i mit Wahrscheinlichkeit p_i in die Menge V_1 und mit Wahrscheinlichkeit $1-p_i$ in die Menge V_2. Es sei nun $v(p_1, \ldots, p_n)$ die erwartete Anzahl an Kanten, die zwischen V_1 und V_2 verlaufen. Es gilt:

$$v(p_1, \ldots, p_n) = \sum_{\{i,j\} \in E} p_i(1 - p_j) + p_j(1 - p_i).$$

Zu Beginn wähle $p_1 = \cdots = p_n = 1/2$, dann ist $v(p_1, \ldots, p_n) = |E|/2$. v ist linear in jedem p_i und effizient zu berechnen. Wir verfahren also wie bei der Derandomisierung erklärt und erhalten am Schluss Werte $p_1, \ldots, p_n \in \{0, 1\}$ mit $v(p_1, \ldots, p_n) \geq |E|/2$. Mit $V_1 := \{i \in V \mid p_i = 1\}$ haben wir also auf effiziente Weise eine Zerlegung berechnet, bei der mindestens $\frac{|E|}{2}$ viele Kanten „kreuzend“ sind.

Beispiel 2: MAXSAT (bzw. MAX-k-SAT): Wir erinnern uns, dass die Variablenbelegung durch randomisiertes Runden berechnet wurde.

- $p_1 = \ldots = p_n = 1/2$, damit hat der randomisierte Algorithmus im Erwartungswert mindestens $(1 - 2^{-k}) \cdot m$ viele Klauseln erfüllt, wobei m die Anzahl der Klauseln ist. (Alle

Klauseln haben Länge k).

- $p_1, \ldots, p_n$ als Lösung eines linearen Programms, damit hat der randomisierte Algorithmus mindestens $(1 - \frac{1}{e}) \cdot \text{opt}$ viele Klauseln erfüllt, wobei opt der Lösungswert des MAXSAT-Problems ist.

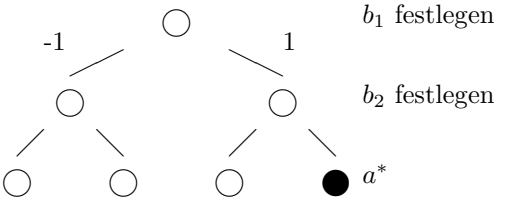
Wir wollen nun die Derandomisierung auf das randomisierte Runden anwenden.
Dafür sei $v(p_1, \ldots, p_n)$ die erwartete Anzahl an Klauseln, die beim randomisierten Runden mit den Parametern $p_1, \ldots, p_n$ erfüllt werden.
Beispielsweise gilt für die Klausel $x_1 \vee x_2 \vee \overline{x_3}$, dass Prob(Klausel erfüllt) $= 1-(1-p_1)(1-p_2)p_3$. Offensichtlich ist auch im allgemeinen Fall die Wahrscheinlichkeit linear in jeder Variablen und effizient berechenbar. Folglich ist auch $v(p_1, \ldots, p_n) = \text{Prob(Klausel 1 erfüllt)} + \cdots + \text{Prob(Klausel } m \text{ erfüllt)}$ linear in jeder Variablen und effizient berechenbar und die Derandomisierung liefert uns $p_1^*, \ldots, p_n^* \in \{0,1\}$ mit $v(p_1^*, \ldots, p_n^*) \geq v(p_1, \ldots, p_n) \geq (1 - \frac{1}{e}) \cdot \text{opt}$.

Beispiel 3: („Kuriosum“: „Manchmal kann man besser derandomisieren.“)
MAXSAT: gesucht ist eine Variablenbelegung, die mindestens die Hälfte aller Klauseln erfüllt.
Beh.: Entweder die Belegung $a_0 = 0 \cdots 0$ oder $a_1 = 1 \cdots 1$ erfüllt die Hälfte aller Klauseln. Der Grund: jede Klausel enthält mindestens ein Literal und dieses Literal wird entweder durch a_0 oder a_1 wahr gemacht. Bsp.: (x bedeutet, dass mit dieser Belegung die Klausel erfüllt ist):

	$0 \cdots 0$	$1 \cdots 1$
$C_1 = x_1 \vee x_2 \vee x_3$		x
$C_2 = x_1 \vee x_2 \vee \overline{x_3}$	x	x
$C_3 = \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$	x	
$\cdots$	?	?
$C_m = \ldots$		

Weil wir mindestens m Kreuze vorliegen haben, gibt es nach dem Schubfachprinzip mindestens eine Spalte, die mindestens $m/2$ viele Kreuze hat.

Beispiel 4: Set Balancing: Gegeben ist eine $n \times n$-Matrix A mit Einträgen aus $\{0,1\}$. Die Aufgabe ist es, einen Vektor $b \in \{-1,1\}^n$ zu finden, so dass $\|A \cdot b\|_\infty$ möglichst klein wird. Wir hatten bereits gezeigt, dass ein Vektor b mit $\|A \cdot b\|_\infty \leq \sqrt{12n \ln n}$ existiert. Wie derandomisiert man das? Dazu sei E_i das Ereignis, dass $|(A \cdot b)_i| > \sqrt{12n \ln n}$ ist. Bei dem Beweis, dass es ein b mit der gewünschten Eigenschaft gibt, hatten wir die Wahrscheinlichkeit des Ereignisses E_i durch $2/n^2$ nach oben abgeschätzt.
Uns interessierte die Versagenswahrscheinlichkeit $\mathbf{Prob}\,(E_1 \cup E_2 \cup \cdots \cup E_n)$, die wir nach oben abgeschätzt haben durch $\mathbf{Prob}\,(E_1) + \cdots + \mathbf{Prob}\,(E_n) \leq 2/n$.
Da wir also nicht mit exakten Wahrscheinlichkeiten rechnen, sondern mit Abschätzungen, sind wir im Prinzip bei der Methode des pessimistischen Schätzers.
Es sei hier erinnert, dass der randomisierte Algorithmus die $b_1, \ldots, b_n$ mit Wahrscheinlichkeit $1/2$ auf -1 oder 1 würfelte. Man stelle die möglichen Variablenbelegungen durch einen Baum dar, z.B.



Knoten legen also die Variablenbelegung partiell fest, z.B. legt der im Baum mit a^* bezeichnete Knoten fest, dass $b_1 = b_2 = 1$ ist. Für einen Knoten a im Baum definieren wir $\text{Prob}(E_i \mid a)$ als die Wahrscheinlichkeit, dass das Ereignis E_i eintritt, nachdem die durch den Knoten a festgelegten

Variablen wie vorgeschrieben fixiert sind. Als Potenzialfunktion für die Derandomisierung wählen wir nun $v(a) := \sum_{i=1}^{n} \text{Prob}(E_i \mid a)$.
Für die Wurzel w des gesamten Baumes wissen wir, dass $v(w) \leq 2/n$ ist.
Für einen Blattknoten a im Baum ist $v(a) \in \{0, 1\}$, wobei $v(a) = 0$ bedeutet, dass das Blatt gut ist, das zugehörige b also die gewünschte Eigenschaft hat.

4.34 Behauptung. $v(a)$ ist effizient berechenbar.

Beweis: Wir zeigen, dass für jedes i und jeden Knoten a die Wahrscheinlichkeit $\mathbf{Prob}\,(E_i \mid a)$ effizient berechenbar ist. Die Behauptung für $v(a)$ ergibt sich daraus.
Wenn wir mit A_i die i-te Zeile der Matrix bezeichnen, dann sei NN die Menge der Positionen in A_i, die den Wert 1 haben, also

$$NN := \{j \mid A_{i,j} = 1\}.$$

Es ist $A_i \cdot b = \sum_{j \in NN} b_j$. Wenn der Knoten a im Baum bedeutet, dass schon $b_1, \ldots, b_k$ zu Konstanten festgelegt sind, dann kann man also das Produkt $A_i \cdot b$ schreiben als

$$\text{const} + \sum_{j \in NN \text{ und } j > k} b_j.$$

Das Ereignis E_i tritt ein, wenn $A_i \cdot b$ nicht in das Intervall $[-\sqrt{12n \ln n}, \sqrt{12n \ln n}]$ fällt. Der Wert von $A_i \cdot b$ hängt offensichtlich von der Anzahl Einsen unter den $r := |\{j \mid j > k \text{ und } j \in NN\}$ noch relevanten b-Positionen ab.
$A_i \cdot b$ fällt offensichtlich genau dann in das Intervall $[-\sqrt{12n \ln n}, \sqrt{12n \ln n}]$, wenn die Anzahl Einsen in ein Intervall [von,bis] fällt, wobei man *von* und *bis* dabei leicht ausrechnen kann.
Wie groß ist also die Wahrscheinlichkeit dafür? Die Wahrscheinlichkeit, dass genau k der Positionen 1 sind, ist offensichtlich $\binom{r}{k} \cdot (1/2)^k \cdot (1/2)^{r-k}$. Die Wahrscheinlichkeit, dass die Anzahl Einsen in das Intervall $[von, bis]$ fällt, ist also

$$\sum_{k=\text{von}}^{\text{bis}} \binom{r}{k} \left(\frac{1}{2}\right)^k \left(\frac{1}{2}\right)^{r-k} = \sum_{k=\text{von}}^{\text{bis}} \binom{r}{k} \left(\frac{1}{2}\right)^r$$

und diese Summe ist effizient berechenbar, $\mathbf{Prob}\,(E_i \mid a)$ ist dann die Komplementärwahrscheinlichkeit davon. □

Für die Wurzel w im Baum ist $v(w) \leq 2/n$. Weiterhin ist

$$\mathbf{Prob}\,(E_i \mid a) = \frac{1}{2}\mathbf{Prob}\,(E_i \mid a_l) + \frac{1}{2}\mathbf{Prob}\,(E_i \mid a_r)\,,$$

wobei a_l der linke und a_r der rechte Nachfolgeknoten ist. Daraus ergibt sich

$$v(a) = \sum_{i=1}^{n} \mathbf{Prob}\,(E_i \mid a) = \frac{1}{2} \cdot v(a_l) + \frac{1}{2} v(a_r).$$

Folglich gilt entweder $v(a_l) \leq v(a)$ oder $v(a_r) \leq v(a)$. Wenn wir also in der Wurzel w starten und jeweils an den Nachfolgerknoten gehen, dessen v–Wert der kleinere der beiden ist, dann kommen wir an einem Blatt b an, in dem

$$v(b) \leq v(w) \leq \frac{2}{n}$$

ist. Da in einem Blatt $v(b) \in \{0, 1\}$ ist, heißt dies hier: $v(b) = 0$. Es gibt somit einen deterministischen Algorithmus, der ein b berechnet mit $\|A \cdot b\|_\infty \leq \sqrt{12n \ln n}$.

Bemerkung: Ein offenes Forschungsproblem: Bekannt ist, dass ein b existiert mit $\|A \cdot b\|_\infty \leq 6\sqrt{n}$. Kann man das Argument derandomisieren?

4.5.2 Reduktion des Wahrscheinlichkeitsraums

Wenn es ohnehin nur polynomiell viele Möglichkeiten gibt, die Zufallsbits zu wählen, so ist ein Derandomisieren effizient möglich. Da der Wahrscheinlichkeitsraum dann nur polynomiell groß ist, kann man ihn mittels vollständiger Suche effizient durchsuchen und so die optimale Wahl der Zufallsbits finden. Man lässt einfach den randomisierten Algorithmus A_{rand} mit allen möglichen Wahlen der Bits $y_1 \dots y_m$ laufen und wählt das beste Ergebnis aus. (Dies ist zum Beispiel dann der Fall, wenn $m = O(\log n)$ ist.)
Manchmal ist es möglich, auch einen exponentiell großen Wahrscheinlichkeitsraum so zu reduzieren, dass man ihn effizient durchsuchen kann. Der besondere Vorteil dieser Methode liegt in der einfachen Parallelisierbarkeit. Im Gegensatz zur Potenzialmethode müssen die Bits nicht sequenziell bestimmt werden. Die Schlüsselidee dazu ist es, statt der Unabhängigkeit der Zufallsbits nur noch k-fache Unabhängigkeit zu fordern, wobei k eine geeignete natürliche Zahl ist. Diese schwächere Forderung führt im allgemeinen aber auch zu einer verschlechterten Güte des daraus resultierenden Algorithmus. Man geht vom Original-Wahrscheinlichkeitsraum (Ω, P) zu einem Wahrscheinlichkeitsraum (Ω', P') über, in dem die Variablen nur k-fach unabhängig sind und die Verteilungsfunktion P' eine Approximation an P darstellt.
Der Einfachheit halber beschäftigen wir uns in diesem Abschnitt nur mit Zufallsvariablen, deren Belegung aus $\{0, 1\}$ ist.

4.35 Definition. Die Zufallsvariablen $X_1, \dots, X_n \in \{0, 1\}$ heißen k-fach unabhängig, wenn jede k–elementige Teilmenge der Zufallsvariablen unabhängig ist, wenn also für alle $1 \le i_1 < i_2 < \dots < i_k \le n$ und alle $(a_1, \dots, a_k) \in \{0, 1\}^k$ gilt, dass

$$\mathbf{Prob}\left((X_{i_1}, X_{i_2}, \dots, X_{i_k}) = (a_1, \dots, a_k)\right) = \mathbf{Prob}\left(X_{i_1} = a_1\right) \cdots \mathbf{Prob}\left(X_{i_k} = a_k\right) \text{ ist.}$$

Wieso kann es günstiger sein, wenn man in einem randomisierten Algorithmus nicht mehr eine vollständige Unabhängigkeit zwischen den Zufallsvariablen benötigt, sondern nur eine etwas schwächere Unabhängigkeit? Um diese Frage zu beantworten, betrachten wir den (ganz extremen) Fall, dass der Algorithmus überhaupt keine Unabhängigkeit zwischen den Zufallsvariablen benötigt und nur ausnutzt, dass jede Zufallsvariable mit Wahrscheinlichkeit 1/2 den Wert 0 annimmt. Eine solche Situation könnte man erreichen, indem man mit Wahrscheinlichkeit 1/2 alle Zufallsvariablen auf 1 setzt und sonst alle Zufallsvariablen auf 0 setzt. Für jede Zufallsvariable gilt dann, dass die Wahrscheinlichkeit, den Wert 1 anzunehmen, genau 1/2 ist. Die Derandomisierung eines solchen Algorithmus wäre recht einfach, denn man müsste den Algorithmus nur für zwei mögliche Belegungen ablaufen lassen, nämlich für die Belegung $(1, 1, \dots, 1)$ und für die Belegung $(0, 0, \dots, 0)$.
Wir zeigen nun, wie man solche kleine Belegungsmengen, auch „kleine Wahrscheinlichkeitsräume“ genannt, konstruieren kann, wenn die Zufallsvariablen k-fach unabhängig sein sollen. Um anschaulich zu bleiben, definieren wir einen (uniformen) Wahrscheinlichkeitsraum wie folgt:

4.36 Definition. Ein Wahrscheinlichkeitsraum ist eine Liste $a_1, \dots, a_T$ von Vektoren, wobei $a_i \in \{0, 1\}^n$ ist für alle $i = 1, \dots, T$.

(Wir haben dabei in der Definition „Liste“ geschrieben, um zu betonen, dass die Vektoren nicht alle verschieden sein müssen.)
Natürlich kann man die $a_1, \dots, a_T$ auch als $T \times n$-Matrix darstellen.
Die Interpretation hinter dieser Definition ist die folgende: Wenn z.B. n=3 ist und wir einen Wahrscheinlichkeitsraum aus T=4 Vektoren gegeben haben, z.B. $(0, 1, 0), (1, 0, 0), (1, 0, 0), (1, 1, 0)$, dann würde man die Belegung der Zufallsvariablen wie folgt auswürfeln: Man würde einen der 4 Vektoren wählen (jeden mit Wahrscheinlichkeit 1/4) und die 3 Zufallsvariablen mit den Werten der entsprechenden Komponenten belegen. Z. B. wäre dann $\mathbf{Prob}\left(X_1 = 0\right) = 1/4$.

Wenn wir n unabhängige Zufallsvariablen benötigen, bei denen jede mit Wahrscheinlichkeit $1/2$ den Wert 1 annimmt, dann könnte man den Wahrscheinlichkeitsraum wählen, der aus allen 2^n Vektoren aus $\{0,1\}^n$ besteht. Dieser Wahrscheinlichkeitsraum hat exponentielle Größe. Beim Derandomisieren würde man 2^n mögliche Belegungen durchprobieren müssen.
Wir zeigen, dass es viel kleinere (polynomiell große) Wahrscheinlichkeitsräume gibt, wenn man nur k-fache Unabhängigkeit der Zufallsvariablen benötigt.

4.37 Satz. *Seien rationale Zahlen $0 \le p_1, \ldots, p_n \le 1$ gegeben. Die Zahl $q \ge n$ sei eine Primzahlpotenz und $k \ge 1$. Dann gibt es einen Wahrscheinlichkeitsraum aus q^k Vektoren, so dass die zugehörigen Zufallsvariablen $X_1, \ldots, X_n$ k-fach unabhängig sind, und es gilt:*

$$|\mathbf{Prob}\,(X_i = 1) - p_i\,| \;\le\; 1/(2q) \quad \textit{für alle } i = 1, \ldots, n.$$

Beweis: Wähle $p_i' = \lceil qp_i - 1/2 \rceil / q$. In Worten: Wir runden p_i auf das nächste Vielfache von $1/q$. Natürlich gilt dann $|p_i' - p_i| \le 1/(2q)$.
Da q eine Primzahlpotenz ist, gibt es einen Körper aus q Elementen, $\mathbf{F}_q = \{k_1, \ldots, k_q\}$. Wir zeichnen n Elemente $f_1, \ldots, f_n$ aus dem Körper aus, z.B. kann man $f_i = k_i$ wählen für $i = 1, \ldots, n$. (Am einfachsten ist es hier vielleicht sogar, sich q als Primzahl vorzustellen, weil dann $\mathbf{F}_q = \mathbf{Z}_q^*$ ist und in diesem Körper kann man mit den üblichen Modulo-Operationen rechnen. Dann wären z.B. auch $f_1 = 0, \ldots, f_n = n-1$.)
Für alle $i = 1, \ldots, n$ wähle man eine beliebige Teilmenge $A_i \subseteq \mathbf{F}_q$ aus, deren Kardinalität $\lceil qp_i - 1/2 \rceil$ ist. Dann gilt $|A_i|/q = p_i'$.
Es gibt q^k viele Polynome vom Grad höchstens $k-1$, deren Koeffizienten aus $\mathbf{F}_q$ sind, wir bezeichnen diese Polynome mit $t_1, \ldots, t_{q^k}$.
Wir definieren die Vektoren a_i des Wahrscheinlichkeitsraums wie folgt:

$$a_{i,j} = \begin{cases} 1, & \text{falls } t_i(f_j) \in A_j, \\ 0 & \text{sonst.} \end{cases}$$

Man kann dies auch so interpretieren, dass man eines der q^k Polynome $c_0 + c_1 x + \cdots + c_{k-1}x^{k-1}$ auswürfelt und anschließend die Belegung der Zufallsvariablen festgelegt ist. Entsprechend sollte sofort klar sein, dass für ein Element f_j und zwei beliebige Elemente $x \ne y$ aus dem Körper gilt, dass

$$\mathbf{Prob}\,(t_i(f_j) = x) = \mathbf{Prob}\,(t_i(f_j) = y)$$

ist. (Dies liegt daran, dass das c_0-Element jedes Körperelement mit der gleichen Wahrscheinlichkeit annimmt.) Somit ist die Wahrscheinlichkeit, dass $X_j = 1$ ist, genau $|A_j|/q = p_j'$.
Zu zeigen bleibt nun noch, dass je k Zufallsvariablen, die durch diesen Wahrscheinlichkeitsraum definiert sind, unabhängig sind. O.B.d.A. betrachten wir die Zufallsvariablen $X_1, \ldots, X_k$. Wie groß ist die Wahrscheinlichkeit $\mathbf{Prob}\,((X_1, \ldots, X_k) = (a_1, \ldots, a_k))$?
Da alle Polynome Grad $k-1$ haben, sind sie durch Festlegung des Funktionswertes an k Stellen eindeutig festgelegt. Betrachten wir beispielhaft die Wahrscheinlichkeit, dass $(X_1, \ldots, X_k) = (1, \ldots, 1)$ ist.
Es gibt $|A_1| \cdots |A_k|$ viele Polynome t, für die für alle $j = 1, \ldots, k$ der Wert $t(f_j) \in A_j$ ist, somit ist

$$\mathbf{Prob}\,((X_1, \ldots, X_k) = (1, \ldots, 1)) = \frac{|A_1| \cdots |A_k|}{q^k} = \frac{|A_1|}{q} \cdots \frac{|A_k|}{q} = \mathbf{Prob}\,(X_1 = 1) \cdots \mathbf{Prob}\,(X_k = 1)\,,$$

was zu zeigen war. Es ist klar, dass eine ähnliche Rechnung gilt, wenn $(a_1, \ldots, a_k) \ne (1, \ldots, 1)$ ist - dort muss man einfach, wenn ein a_i Null ist, die Menge A_i durch die Komplementmenge $\mathbf{F}_q \setminus A_i$ ersetzen. □

Beispiel für einen kleinen Wahrscheinlichkeitsraum Für unser Beispiel wählen wir $q = n = 7$ sowie $k = 2$ und haben dann mit 49 Polynomen zu tun, nämlich den Polynomen $t_1 = 0 + 0 \cdot x, t_2 = 0 + 1 \cdot x, \ldots, t_{49} = 6 + 6 \cdot x$. Außerdem wählen wir $f_1 = 0, \ldots, f_7 = 6$ und alle $A_i := \{4, 5, 6\}$. Damit erreichen wir, dass $\mathbf{Prob}\,(X_i = 1) = 3/7 = 21/49$ ist für alle i.

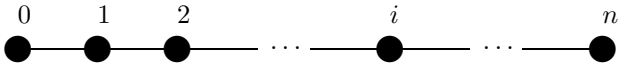
Z.B. erhalten wir $t_1(f_1) = 0, \ldots, t_1(f_7) = 0$ und somit als ersten Vektor des Wahrscheinlichkeitsraums den Vektor $(0,0,0,0,0,0,0)$. Da $t_2(f_1) = 0, t_2(f_2) = 1, \ldots, t_2(f_7) = 6$ und $t_3(f_1) = 0, t_3(f_2) = 2, t_3(f_3) = 4, t_3(f_4) = 6, \ldots, t_3(f_7) = 5$, ist der zweite Vektor $(0,0,0,0,1,1,1)$ und der dritte $(0,0,1,1,0,0,1)$. Wenn man so fortfährt, erhält man die 49 Vektoren

```
0000000   0100101   0001110   1100100   1001100
0000111   0101001   0000000   1001010   1100001
0011001   0110010   0111000   1010100   1111111
0010101   0011100   0100110   1001001   1000011
0101010   0000000   0101010   1000011   1001001
0100110   0011100   0010101   1111111   1010100
0111000   0110010   0011001   1100001   1001010
0000000   0101001   0000111   1001100   1100100
0001110   0100101   1111111   1010010   1110000
0010011   0010011   1110000   1010010
```

(Wie man sieht, kommen manche Vektoren mehrfach vor.) Man überzeuge sich, dass in jeder Position (über alle Vektoren betrachtet) genau 21 Einsen vorkommen. Dies heißt, dass $\mathbf{Prob}(X_i = 1) = 21/49$ ist. Man überzeuge sich auch, dass für jedes Paar $i \neq j$ von Positionen gilt, dass die Wahrscheinlichkeit, ein Muster $(0,0)$ in diesen Positionen zu sehen, gleich $28/49 \cdot 28/49 = 16/49$ ist, ein Muster $(0,1)$ oder $(1,0)$ zu sehen, gleich $21/49 \cdot 28/49 = 12/49$ ist und die Wahrscheinlichkeit, das Muster $(1,1)$ zu sehen, gleich $21/49 \cdot 21/49 = 9/49$ ist. Dies entspricht der 2-fachen Unabhängigkeit.
(Beispielsweise haben für $i = 1, j = 2$ genau 9 der 49 Vektoren in den ersten beiden Positionen das Muster $(1,1)$.)

4.6 Random Walks und Markovketten

Wir beginnen dieses Kapitel mit einem einleitenden Beispiel für einen random walk.



Wir führen die folgende Prozedur durch.

```
pos:=i;
repeat
  if pos=0 then STOP;
  else
    if pos=n then pos:=n-1;
    else mit Wahrscheinlichkeit je 1/2:  pos:=pos+1; oder pos:=pos-1;
until false
```

Wie lange dauert es, bis dieser „Spaziergang“ terminiert?

4.38 Definition. Sei E_i die erwartete Anzahl an Schritten, die es dauert, bis Position 0 erreicht wird, wenn wir in Position i starten.

4.39 Satz. $E_i = i \cdot (2n - i)$. *Beispiel:* $E_1 = 2n - 1$ *und* $E_n = n^2$.

Beweis: $E_0 = 0$, $E_n = 1 + E_{n-1}$ und $E_i = \frac{1}{2}E_{i-1} + \frac{1}{2}E_{i+1} + 1$ für $i \in \{1, \ldots, n-1\}$. Damit wir diese Gleichungen aufstellen dürfen, müssten wir vorweg $E_i < \infty$ nachweisen, aber diese kleine

Nachlässigkeit gönnen wir uns hier. Wenn wir die letzte Gleichung umstellen, erhalten wir, dass auch

$$\frac{1}{2}(E_i - E_{i-1}) = \frac{1}{2}(E_{i+1} - E_i) + 1$$

wahr ist. Mit der Abkürzung $D_i := E_i - E_{i-1}$ wird daraus $D_i/2 = 1 + D_{i+1}/2$ bzw. $D_i = D_{i+1} + 2$. Da $D_n = E_n - E_{n-1} = 1$, ergibt sich

$$D_n = 1, D_{n-1} = 3, D_{n-2} = 5, \ldots, D_1 = 2(n-1) + 1.$$

Mit Hilfe der D_i-Werte können wir nun die E_i-Werte ausrechnen: Es ist

$$\begin{aligned} D_1 + D_2 + \cdots + D_i &= (E_1 - E_0) + (E_2 - E_1) + \cdots + (E_i - E_{i-1}) \\ &= E_1 + E_2 + \cdots + E_i - E_0 - E_1 - \cdots - E_{i-1} \\ &= E_i - E_0 = E_i. \end{aligned}$$

Also ist $E_i = (2n-1) + (2n-3) + \cdots = 2ni - (1 + 3 + \cdots) = 2ni - i^2$. □

4.6.1 Markovketten

4.40 Definition. Eine endliche **Markovkette** ist ein stochastischer Prozess mit:

- diskreter Zeit,
- endlicher Zustandsmenge $S = \{1, \ldots, n\}$,
- Werten p_{ij} für alle $i, j \in S$, die die Wahrscheinlichkeiten für den Übergang vom Zustand i in den Zustand j angeben, zusammengefasst in der Matrix $P = (p_{ij})$.

In dieser Vorlesung werden wir bis auf Weiteres nur endliche Markovketten behandeln, daher meinen wir im folgenden mit „Markovkette“ immer eine *endliche* Markovkette.

Die Matrix P hat aufgrund ihrer Definition die Eigenschaft, dass alle ihre Zeilensummen gleich 1 sind.
Die Matrix P ist zeitunabhängig: Definiere Zufallsvariablen X_t, die angeben, in welchem Zustand sich die Markovkette zum Zeitpunkt t befindet. Dann gilt:

$$\mathbf{Prob}\,(X_{t+1} = j \mid X_0 = i_0, X_1 = i_1, \ldots, X_t = i_t = i) = \mathbf{Prob}\,(X_{t+1} = j \mid X_t = i) = p_{ij}.$$

Diese Eigenschaft heißt auch *Gedächtnislosigkeit.*
Schließlich benötigen wir noch einen Startzustand bzw. eine Wahrscheinlichkeitsverteilung

$$q^{(0)} = (q_1^{(0)}, \ldots, q_n^{(0)}),$$

die angibt, dass wir mit Wahrscheinlichkeit $q_i^{(0)}$ in Zustand i starten. Ein wichtiger Spezialfall ist der, bei dem genau ein Startzustand festgelegt ist, hierbei ist $q_i^{(0)} = 1$ für ein i.
Die Verteilung zum Zeitpunkt $t = 1$ ergibt sich als Matrix-Vektor-Multiplikation für Zeilenvektoren

$$q^{(1)} = q^{(0)} \cdot P$$

und wegen der Gedächtnislosigkeit gilt allgemein

$$q^{(t)} = q^{(t-1)} \cdot P, \text{ also } q^{(t)} = q^{(0)} \cdot P^t.$$

Wir definieren im folgenden einige wichtige Begriffe, die bei der Analyse von Markovketten von Bedeutung sind:

4.41 Definition. Eine Zustandsübergangsmatrix heißt **doppelt stochastisch**, wenn alle ihre Spaltensummen gleich 1 sind.

Eine Verteilung π über S heißt **stationär**, wenn $\pi \cdot P = \pi$ gilt.

Mit dem Parameter $r_{ij}^{(t)}$ bezeichnen wir die Wahrscheinlichkeit, dass bei Start in Zustand i der Zustand j das nächste Mal nach genau t Schritten erreicht wird:

$$r_{ij}^{(t)} := \mathbf{Prob}\left(X_t = j \ \wedge \ \ (\forall s \in [1, t-1] : X_s \neq j) \mid X_0 = i\right).$$

Mit dem Parameter $f_{ij} := \sum_{t \geq 1} r_{ij}^{(t)}$ bezeichnen wir die Wahrscheinlichkeit, bei Start in Zustand i irgendwann den Zustand j zu erreichen. Die Zahl f_{ii} nennen wir auch „Rückkehrwahrscheinlichkeit“.

Für $i \neq j$ definieren wir $h_{ij} := \begin{cases} \sum_{t \geq 1} t \cdot r_{ij}^{(t)} & \text{falls } f_{ij} = 1 \\ \infty & \text{falls } f_{ij} < 1 \end{cases}$
Diese Zahl misst die erwartete Anzahl an Schritten, die es dauert, bis man in i startend j zum ersten Mal erreicht hat.
Vorsicht: Aus $f_{ij} = 1$ folgt in der Regel nicht $h_{ij} < \infty$! h_{ij} wird auch mittlere erste Besuchszeit genannt.

Wie definiert man nun h_{ii}? In der Literatur wird h_{ij} für den Fall $i = j$ genau so definiert wie oben im Fall $i \neq j$. Dies hat aber einen Nachteil, den ich hier beschreiben möchte.
Wenn man von einem Zustand i in mehreren Schritten zu einem Zustand j geht, so beginnt dieser Weg mit einem ersten Schritt, zum Beispiel entlang einer Kante von i zu einem Zustand k. Daher kann man für die erwartete Schrittzahl h_{ij} von Zustand i nach Zustand j eine Gleichung der Form

$$h_{ij} = 1 + \sum_{k \text{ Zustand}} p_{ik} \cdot h_{kj}$$

aufstellen. (Die 1 misst dabei den ersten Schritt.)
Für die Gültigkeit dieser Gleichung braucht man dabei aber, dass $h_{jj} = 0$ ist, denn wenn man im ersten Schritt von Knoten i zu Knoten $k = j$ geht, dann ist man schon am Ziel und muss nicht erst noch von Knoten j wieder zu Knoten j zurückkehren. Aus diesem Grund definieren wir $h_{ii} := 0$ und

$$h_{ii}^* := \begin{cases} \sum_{t \geq 1} t \cdot r_{ii}^{(t)} & \text{falls } f_{ii} = 1 \\ \infty & \text{falls } f_{ii} < 1 \end{cases},$$

also genau so, wie es für den Fall $i \neq j$ war.

Ein Zustand i heißt **transient**, wenn $f_{ii} < 1$ ist und **persistent** sonst.

Ein Zustand i heißt **ergodisch**, wenn $h_{ii}^* < \infty$.

Wir wollen nun die **Periodizität** eines Zustandes i definieren. Leider gibt es in der Literatur verschiedene Definitionen, die aber für die meisten Markovketten zusammenfallen. Wir entscheiden uns hier für eine dieser Definitionen.
Wenn man den Zustand i als Startpunkt wählt, so sei $T(i)$ die Menge aller Zeitpunkte, zu denen sich die Markovkette mit Wahrscheinlichkeit größer 0 in Zustand i aufhalten kann. Formaler: Sei die Startverteilung $q^{(0)}$ gleich dem i-ten Einheitsvektor. Dann sei

$$T(i) := \{t \mid q_i^{(t)} > 0\}.$$

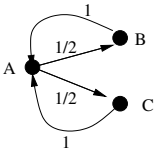
Die Periodizität des Zustands i ist nun der größte gemeinsame Teiler aller Zahlen in $T(i)$.

Ein Zustand i heißt **aperiodisch**, wenn seine Periodizität gleich 1 ist.

Eine Markovkette heißt **aperiodisch**, wenn alle ihre Zustände aperiodisch sind.

Eine Markovkette wird kanonisch durch einen Zustandsübergangsgraphen dargestellt, bei dem die Knoten den Zuständen entsprechen und Kanten mit den Übergangswahrscheinlichkeiten beschriftet sind. Dabei werden Kanten mit der Wahrscheinlichkeit 0 weggelassen.
Eine Markovkette heißt **irreduzibel**, wenn alle Zustände in derselben starken Zusammenhangskomponente des Zustandsübergangsgraphen liegen.

4.42 Beispiel. Die Periodizidät des Zustands A in der folgenden Markovkette ist 2, da $T(A) = \{0, 2, 4, 6, \ldots\}$ ist.



4.43 Satz (Fundamentales Theorem über Markovketten)**.**

Für eine irreduzible, aperiodische Markovkette mit Zustandsmenge S gilt:

1. Alle Zustände i der Markovkette sind ergodisch, haben also eine endliche erwartete Rückkehrzeit h^*_{ii}.
2. Es existiert eine eindeutige stationäre Verteilung $\pi = (\pi_1, \ldots, \pi_n)$ mit $\pi_i > 0$ für alle Zustände i.
3. Für alle Zustände i gilt $h^*_{ii} = (\pi_i)^{-1}$.
4. Für alle Zustände i gilt: Wenn $N(i,t)$ die erwartete Anzahl der Besuche von Zustand i in t aufeinander folgenden Schritten bezeichnet, dann ist

$$\lim_{t\to\infty} \frac{N(i,t)}{t} = \pi_i.$$

Dieses fundamentale Theorem werden wir in der Vorlesung zwar benutzen, aber nicht beweisen.

4.44 Lemma. *Für eine irreduzible, aperiodische Markovkette mit doppelt stochastischer Zustandsübergangsmatrix gilt: Ihre (eindeutige) stationäre Verteilung ist die Gleichverteilung auf der Zustandsmenge S.*

Beweis: Sei $n := |S|$. Es reicht, zu zeigen, dass für $\pi = (1/n, \ldots, 1/n)$ gilt, dass $\pi \cdot P = \pi$ ist, die Eindeutigkeit folgt aus dem fundamentalen Theorem. Für jede Matrix P gilt

$$\pi \cdot P = (1/n) \cdot (1, 1, \ldots, 1) \cdot P = (1/n) \cdot (s_1, \ldots, s_n),$$

wobei s_i die i-te Spaltensumme ist. Bei einer doppelt stochastischen Zustandsübergangsmatrix ist $s_i = 1$ für alle i, also $\pi \cdot P = (1/n, \ldots, 1/n)$. □

4.6.2 Random Walks auf Graphen

Wenn G ein ungerichteter Graph ist, dann kann man, wie in der folgenden Definition beschrieben, eine Markovkette definieren, die einen Random Walk auf G beschreibt.

4.45 Definition. Mit der Wahl von $S := V = \{1, \ldots, n\}$ sowie

$$p_{uv} := \begin{cases} 0 & \{u, v\} \notin E \\ 1/grad(u) & \text{sonst} \end{cases}$$

definiert der gegebene Graph eine Markovkette $M(G)$.

Klar ist, dass $M(G)$ genau dann irreduzibel ist, wenn G zusammenhängend ist. Außerdem gilt: $M(G)$ ist genau dann aperiodisch, wenn G nicht bipartit ist. Für den Graphen aus genau einem Knoten ist dies trivial, für andere Graphen sieht man es wie folgt ein: Ist der Graph bipartit, dann kann man immer nur zu geraden Zeitpunkten im Startzustand i sein. Ist der Graph nicht bipartit, dann enthält er einen Kreis ungerader Länge. Somit kann man von jedem Zustand i auf einem Weg ungerader Länge zu i zurückkehren: Man steuere von i aus den Kreis an, dies geschehe mit t Kanten. Dann durchlaufe man den Kreis ungerader Länge einmal und kehre über die t Kanten wieder zu i zurück. Die Gesamtzahl benutzer Kanten ist offensichtlich ungerade.
Andererseits gibt es vom Zustand i aus einen Weg der Länge 2 zum Zustand i zurück: Da G zusammenhängend ist und mindestens zwei Knoten enthält, kann man von i aus eine Kante verlassen und über die gleiche Kante sofort zurückkehren.
Damit ist der größte gemeinsame Teiler 1 und die Markovkette $M(G)$ aperiodisch.

4.46 Beispiel. Wenn G ein d-regulärer Graph ist, das heißt, jeder Knoten in G hat Grad genau d, dann ist die Zustandsübergangsmatrix der zugeordneten Markovkette $M(G)$ doppelt stochastisch.

4.47 Lemma. *Sei G ein ungerichteter, zusammenhängender Graph, der nicht bipartit ist. Für die durch G gegebene Markovkette gilt: Es existiert eine eindeutige stationäre Verteilung $\pi = (\pi_1, \ldots, \pi_n)$ und es gilt für alle $v \in V$:*

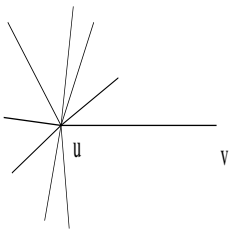
$$\pi_v = \frac{grad(v)}{2|E|} \text{ und } h^*_{vv} = \frac{2|E|}{grad(v)}.$$

Beweis: Es reicht, $\pi_v = \frac{grad(v)}{2|E|}$ zu zeigen, da die Aussage für h^*_{vv} sich dann aus dem fundamentalen Theorem ergibt.
Da G zusammenhängend ist, ist die Markovkette irreduzibel. Da G nicht bipartit ist, ist sie auch aperiodisch. Die Existenz und Eindeutigkeit der stationären Verteilung folgt aus dem fundamentalen Theorem. Sei $v \in V$. Wir bezeichnen im folgenden die Menge der Nachbarn von v im Graphen als $\Gamma(v)$. Es gilt:

$$(\pi P)_v = \pi_1 \cdot p_{1v} + \cdots + \pi_n \cdot p_{nv} = \sum_{u \in V} \pi_u \cdot p_{uv} = \sum_{u \in \Gamma(v)} \frac{grad(u)}{2|E|} \cdot \frac{1}{grad(u)} = \sum_{u \in \Gamma(v)} \frac{1}{2|E|} = \frac{grad(v)}{2|E|}.$$

□

In der Regel interessiert uns jedoch eher die erwartete Anzahl an Schritten, um von einem Knoten u zu einem Knoten $v \neq u$ zu gelangen. Wie das folgende Beispiel zeigt, ist diese Zahl im Allgemeinen nicht symmetrisch, d.h. es gilt $h_{uv} \neq h_{vu}$:



In diesem Graph ist $h_{uv} \geq n-1$, aber $h_{vu} = 1$.
(Der Graph ist zwar bipartit, aber die Zahlen $h_{u,v}$ sind ja auch für beliebige Markovketten wohldefiniert.)
Wir betrachten nun die so genannte **commute**-Zeit

$$C_{uv} := h_{uv} + h_{vu} = C_{vu},$$

die offensichtlich symmetrisch ist.

4.48 Lemma. *Sei G zusammenhängend und nicht bipartit. Für die commute-Zeit des durch G gegebenen Random Walks gilt:*

$$\forall\ \{u,v\} \in E: \quad C_{uv} \leq 2|E|.$$

Beachte: *Die Bedingung $\{u,v\} \in E$ ist wichtig in der Voraussetzung, da für Knoten u, v, zwischen denen es* keine *Kante gibt, die Aussage nicht notwendigerweise gilt.*

Beweis: Wir definieren zur Analyse eine neue Markovkette, deren Zustände angeben, welcher Kante im Random Walk auf G zuletzt (und in welcher Richtung) gefolgt wurde: S enthält für jede Kante $e = \{u,v\} \in E$ die beiden Zustände (u,v) und (v,u).
Ihre Zustandsübergangsmatrix Q hat dann die Gestalt $Q_{(u,v)(v,w)} = p_{vw} = 1/grad(v)$ und $Q_{(x,y)(v,w)} = 0$ für $y \neq v$.
Wir zeigen zunächst, dass Q doppelt stochastisch ist: Betrachte einen festen Zustand (v,w) der neuen Markovkette. Die Spaltensumme der zugehörigen Spalte ist:

$$\begin{aligned}
\sum_{x\in V, y\in\Gamma(x)} Q_{(x,y),(v,w)} &= \sum_{u\in\Gamma(v)} Q_{(u,v),(v,w)} \\
&= \sum_{u\in\Gamma(v)} \frac{1}{grad(v)} \\
&= grad(v) \cdot \frac{1}{grad(v)} \\
&= 1.
\end{aligned}$$

Damit hat die neue Markovkette nach Lemma 4.44 die Gleichverteilung als stationäre Verteilung π. Da wir $2|E|$ viele Zustände vorliegen haben, ist also $\pi_{(u,v)} = 1/(2|E|)$. Weiter gilt dann nach dem fundamentalen Theorem $h^*_{(u,v),(u,v)} = 1/\pi_{u,v} = 2|E|$. Das fundamentale Theorem ist anwendbar, da die neue Markovkette, wie man sich leicht überlegt, sowohl irreduzibel als auch nicht bipartit ist, weil die ursprüngliche Markovkette $M(G)$ schon diese Eigenschaften hatte.
Mit $h^*_{(v,u),(v,u)}$ misst man, wie lange es im Erwartungswert dauert, bis man, startend mit der Kante $v \to u$, wieder über die Kante $v \to u$ nach u gelangt. Wenn dieses Ereignis eingetreten ist, dann ist auch folgendes Ereignis eingetreten:
Startend mit der Kante $v \to u$ ist man von u nach v gelaufen und dort von v wieder nach u zurück.
Da wir eine Markovkette vorliegen haben, ist die erwartete Zeit, von u nach v und von dort wieder zu u zu gelangen, aber vollkommen unabhängig davon, wie wir zu Beginn in den Knoten u gelangt sind. Deshalb können wir nun abschätzen:

$$h_{u,v} + h_{v,u} \leq h^*_{(v,u),(v,u)} = 2|E|.$$

□

Manchmal interessiert uns nicht so sehr, wie lange es dauert, um von einem ausgezeichneten Knoten zu einem anderen Knoten zu gelangen, sondern wir möchten wissen, wie lang es dauert, bis man alle Knoten des Graphen mindestens einmal aufgesucht hat. Daher definieren wir nun mit $\mathcal{C}_u(G)$ die erwartete Anzahl an Schritten, die ein in u gestarteter random walk macht, bis alle Knoten des Graphen G besucht sind. Die Überdeckungszeit („Cover time“) des Graphen ist dann definiert durch

$$\mathcal{C}(G) = \max_{u \in V} \mathcal{C}_u(G).$$

Hier eine Übersicht über die Namen der Werte:

Parameter	englischer Ausdruck	deutscher Ausdruck
u, v	vertex	Knoten
$h_{u,v}$	hitting time	mittlere erste Besuchszeit
$C_{u,v}$	commute time ($h_{u,v} + h_{v,u}$)	Rückkehrzeit (über v)
$\mathcal{C}(G)$	Cover time	Überdeckungszeit

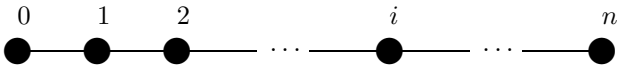
Frage: Kann man die Werte systematisch berechnen?
Dies führt uns zu *elektrischen Netzwerken*. Wir interpretieren einen zusammenhängenden Graphen G wie folgt als elektrisches Netzwerk:
Eine Kante im Graphen erhält den Widerstand 1. (Wir benutzen hier keine Einheiten). Aus dem Ohmschen Gesetz $R = U/I$ folgt somit $U = I$. Es soll außerdem die Kirchhoffregel gelten, die besagt, dass die Summe der einfließenden Ströme gleich der der ausfließenden Ströme ist. Wenn man einfließenden Strom negativ und ausfließenden Strom positiv bewertet, ist die dazu äquivalente Aussage, dass die Summe der Ströme gleich 0 ist.
Die Spannung zwischen zwei Knoten ist gleich der Differenz der entsprechenden Potenziale. Zum Netzwerk gehört auch, dass wir Stromquellen vorgeben. Dies geschieht so, dass wir in bestimmte Knoten Strom einspeisen und an anderen Knoten Strom entnehmen. (Damit die Kirchhoffregeln gelten können, muss die Gesamtsumme Null sein.)
Es stellen sich Potenziale und somit Ströme und Spannungen im Netzwerk ein. Zwischen zwei Knoten mit den Potenzialen p_1 und p_2 fließt ein Strom der Größe $|p_2 - p_1|$. Um die Richtung festzulegen, sagen wir, dass Strom immer vom größeren zum kleineren Potenzial fließt. Wir überlegen nun, wie wir die Potenziale berechnen können.
Zuerst legen wir willkürlich für einen Knoten v dessen Potenzial als $pot_v = 0$ fest. (Dies entspricht einer Erdung des Knotens v.)
Die anderen Gleichungen ergeben sich aus der Kirchhoffregel. Wenn wir mit S_u die Stromstärke bezeichnen, die wir in Knoten u einspeisen, dann muss gelten:

$$\sum_{x \in \Gamma(u)} (pot_u - pot_x) = S_u.$$

Dieses Gleichungssystem (mit der Bedingung $pot_v = 0$) hat eine eindeutige Lösung! Dass es maximal eine Lösung gibt, beweisen wir im Anhang. Dass es mindestens eine Lösung gibt weiß man ja aus der Elektrotechniki, denn es stellen sich ja immer bestimmte Spannungen und Ströme ein.
Wir zeigen nun, dass wir die mittleren ersten Besuchszeiten mit Hilfe der Potenziale ausrechnen können. Zunächst brauchen wir eine Definition:

4.49 Definition. Der *effektive Widerstand* $R_{u,v}$ ist der Betrag der Spannung, die sich zwischen u und v einstellt, wenn wir in Knoten u den Strom 1 einspeisen und im Knoten v den Strom 1 entnehmen.

4.50 Beispiel. Wir betrachten folgende Kette:



Wenn wir in Knoten n den Strom 1 einspeisen und in Knoten 0 entnehmen, dann kann man sich leicht überlegen, dass mit der Wahl von $pot_i = i$ die Kirchhoffregeln eingehalten sind und damit der Betrag der Spannung zwischen Knoten 0 und Knoten n den Wert n hat.

Bei der Analyse der Überdeckungszeiten $C_{u,v}$ erweist es sich als nützlich, auf bestimmte Art und Weise in das zum Graphen G konstruierte Netzwerk Ströme einzuspeisen und zu entnehmen.

4.51 Definition. Für einen Knoten $v \in V$ definieren wir zunächst das Netzwerk $N_v^*(G)$, das sich aus G wie folgt ergibt:

- speise in jeden Knoten $x \neq v$ den Strom $d(x)$ ein.
- entnimm an Knoten v den Strom $2 \cdot |E| - d(v)$.

Das Netzwerk $N_v(G)$ ist analog zu $N_v^*(G)$ definiert, nur sind hier die Begriffe „einspeisen“ und „entnehmen“ vertauscht.

4.52 Satz. *Im Netzwerk $N_v^*(G)$ ergeben sich zwischen je zwei Knoten $x \neq y$ Potenzialdifferenzen $\Phi_{x,y}^{N_v^*} := |pot_x - pot_y|$. Für alle $u \neq v \in V$ gilt:*

$$h_{u,v} = \Phi_{u,v}^{N_v^*}.$$

Beweis: Betrachte einen beliebigen Knoten $u \in V$ mit $u \neq v$ im Netzwerk $N_v^*(G)$.
Wenn $w_1, w_2, \ldots, w_r$ seine Nachbarn sind, dann gilt für u (nach Kirchhoff):

$$\begin{aligned} -d(u) + (pot_u - pot_{w_1}) + (pot_u - pot_{w_2}) + \cdots + (pot_u - pot_{w_r}) &= 0, \text{ also} \\ -d(u) + pot_u \cdot d(u) - pot_{w_1} - pot_{w_2} - \cdots - pot_{w_r} &= 0, \text{ also} \\ d(u) + pot_{w_1} + pot_{w_2} + \cdots + pot_{w_r} &= pot_u \cdot d(u) \end{aligned}$$

Indem wir durch $d(u)$ teilen, erhalten wir

$$pot_u = 1 + \sum_{x \in \Gamma(u)} \frac{pot_x}{d(u)}.$$

Nun beachten wir, dass wir jede Zahl C schreiben können als $C = \sum_{x \in \Gamma(u)} \frac{1}{d(u)} \cdot C$. Daher können wir die letzte Gleichung wie folgt umformen:

$$\begin{aligned} pot_u - pot_v &= (1 - pot_v) + \sum_{x \in \Gamma(u)} \frac{pot_x}{d(u)} \\ &= \sum_{x \in \Gamma(u)} \frac{pot_x + 1 - pot_v}{d(u)}, \end{aligned}$$

und somit mit der Abkürzung $\Phi_{a,b} = pot_a - pot_b$:

$$\Phi_{u,v} = \sum_{x \in \Gamma(u)} \frac{1 + \Phi_{x,v}}{d(u)}.$$

Betrachte nun, wie man die $h_{u,v}$ berechnen kann:

$$h_{u,v} = \sum_{x \in \Gamma(u)} \frac{1}{d(u)} \left(1 + h_{x,v}\right).$$

Der Grund dafür ist, dass ein random walk von u nach v mit einer Kante von u zu einem Nachbarn x von u beginnt (und zwar mit der Wahrscheinlichkeit $1/d(u)$) und von x nach v fortgesetzt werden muss. $\Phi_{u,v}$ und $h_{u,v}$ sind beides Lösungen des gleichen linearen Gleichungssystems. Da dieses Gleichungssystem eine eindeutige Lösung hat, folgt $h_{u,v} = \Phi_{u,v}$. □

4.53 Satz. *Sei G zusammenhängend. Dann gilt:*

$$\forall u, v \in V, u \neq v : C_{u,v} = 2 \cdot |E| \cdot R_{u,v}$$

Beweis: Nach Definition ist $C_{u,v} = h_{u,v} + h_{v,u}$ und wegen Satz 4.52 ist $h_{u,v} = \Phi_{u,v}^{N_v^*}$. Hieraus ergibt sich

$$h_{v,u} = \Phi_{v,u}^{N_u^*} = -\Phi_{v,u}^{N_u} = \Phi_{u,v}^{N_u}.$$

Da sich Netzwerke (bzw. die Spannungen und Ströme in ihnen) additiv verhalten, gilt

$$h_{u,v} + h_{v,u} = \Phi_{u,v}^{N_v^*} + \Phi_{u,v}^{N_u}.$$

Also folgt zuletzt $C_{u,v} =$ Potenzialdifferenz in $(N_v^* + N_u) = R_{u,v} \cdot 2 \cdot |E|$. Damit dies gilt, muss man sich nur noch überlegen, dass die Überlagerung der beiden Netzwerke das „$2|E|$-fache“ des ursprünglich definierten Netzwerks ergibt. Dies ist aber relativ leicht nachzuvollziehen. □

4.54 Satz. *Sei G ein zusammenhängender Graph und $u \neq v$ zwei Knoten aus G. Wenn $dist(u, v)$ die Länge eines kürzesten Wegs von u nach v angibt, dann gilt:*

$$R_{u,v} \leq dist(u, v) \text{ und somit } C_{u,v} \leq 2|E| \cdot dist(u, v).$$

Den Beweis wollen wir nur skizzieren: Wenn man nur einen Weg (der dann der kürzeste wäre) zwischen u und v hätte, so wäre der effektive Widerstand genau gleich der Länge des Weges, es kann eventuell aber zusätzlich noch Strom von u nach v über andere Wege fließen, wodurch der effektive Widerstand höchstens kleiner wird.
Mit diesen beiden Sätzen kann ein alternativer Beweis zum Lemma 4.48 erbracht werden: Wenn u und v zwei Knoten sind, zwischen denen es eine Kante gibt, dann ist

$$C_{u,v} \leq 2 \cdot |E|.$$

Der Grund ist, dass die Existenz einer Kante zwischen u und v bedeutet, dass die Länge des kürzesten Weges zwischen den beiden genau 1 ist.
Auch erhält man folgendes Korollar:

4.55 Satz (Korollar)**.** *Sei G zusammenhängend. Dann ist für alle $u \neq v$:*

a) $C_{u,v} \leq 2 \cdot (n-1) \cdot |E|$.

b) $C_{u,v} < n^3$.

Dabei folgt a) aus der Tatsache, dass in einem zusammenhängenden Graphen der kürzeste Weg zwischen u und v Länge höchstens $n-1$ hat und b) folgt aus a) und der Tatsache $|E| \leq n \cdot (n-1)/2$. Wenden wir uns nun mit dem folgenden Satz den Überdeckungszeiten zu.

4.56 Satz. *Sei G ein zusammenhängender Graph. Dann gilt $\mathcal{C}(G) \leq 2 \cdot |E| \cdot (n-1)$.*

Beweis: Sei T ein Spannbaum von G und u ein Knoten von G. Es gibt einen Durchlauf D durch den Baum T, der in u startet und jede Kante genau zweimal benutzt, und zwar in jeder Richtung einmal. (Beweisidee hierzu: Starte einen DFS-Durchlauf in u. Wenn DFS die Kante (u, v) betrachtet, benutze diese Kante auch in D und wenn nach Abarbeitung des Knotens v DFS zum Knoten u zurückkehrt, benutze in D die Kante (v, u).)
Der Durchlauf hat $2n-2$ Kanten, sei er beschrieben durch die Knotenfolge $u = v_0 \to v_1 \to \cdots \to v_{2n-2} = u$. Dann können wir offensichtlich abschätzen:

$$\mathcal{C}(G) \leq \sum_{j=0}^{2n-3} h_{v_j, v_{j+1}}.$$

Da jede Kante von T in jeder Richtung genau einmal durchlaufen wird, kann man dies auch schreiben als

$$\sum_{e=\{u,w\} \in T} (h_{u,w} + h_{w,u}) = \sum_{e=\{u,w\} \in T} C_{u,w}.$$

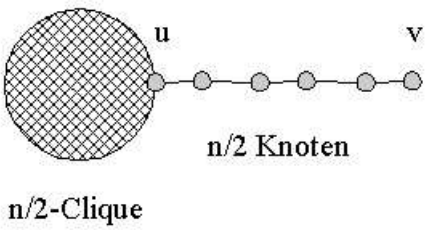
Nach Lemma 4.48 gilt $C_{u,w} \leq 2|E|$ für $\{u, w\} \in E$. Da für jedes betrachtete Knotenpaar $\{u, w\}$ in der obigen Summe die Bedingung $\{u, w\} \in E$ stets erfüllt ist, folgt

$$\sum_{e=\{u,w\} \in T} C_{u,w} \leq \sum_{e=\{u,w\} \in T} 2|E| = 2|E|(n-1).$$

□

Wir betrachten nun ein Beispiel:

4.57 Definition. Sei n gerade. Der Lollipop-Graph L_n mit n Knoten besteht aus einer $n/2$-Clique und einer Kette aus $n/2$ Knoten, wie im folgenden Bild skizziert.



Wir wollen für die beiden eingezeichneten Knoten u und v die Werte h_{uv} und h_{vu} bestimmen. Die Tatsache, dass $h_{vu} = \Theta(n^2)$ ist, ergibt sich aus der Tatsache, dass wir für die Kette, die wir in Satz 4.39 analysiert hatten, eine erwartete Laufzeit von $\Theta(n^2)$ bekommen haben. Dieses Resultat überträgt sich hier direkt, denn ein random walk, der in v startet und dann endet, wenn er sich zum ersten Mal in u befindet, „bekommt nicht mit, ob sich jenseits des Knotens u noch weitere Knoten befinden."
Wenn man h_{uv} betrachtet, also in Knoten u startet, dann muss man anders argumentieren:

Wir benutzen Satz 4.52 und entnehmen in Knoten v genau $2|E|-1$ Stromeinheiten, speisen in die Knoten zwischen u und v jeweils 2 Stromeinheiten ein und analysieren die Potenzialdifferenzen, die sich einstellen. Der Einfachheit halber „erden“ wir v, setzen also $pot_v := 0$.
Um besser über die Knoten reden zu können, nummerieren wir sie von v aus gesehen durch und bezeichnen mit v_0 den Knoten v, mit v_1 den linken Nachbar von v_0 und so weiter bis zum Knoten u.
Offensichtlich muss wegen der am Knoten v geltenden Kirchhoffregel von v_1 nach v_0 der Strom $2|E|-1$ fließen, also muss v_1 das Potenzial $2|E|-1$ haben.
Da in v_1 genau 2 Einheiten von außen eingespeist werden, muss von v_2 nach v_1 Strom $2|E|-3$ fließen, somit hat v_2 das Potenzial $2|E|-1+2|E|-3$. Entsprechend kann man für Knoten v_i nachweisen, dass er Potenzial $i \cdot 2|E| - 1 - 3 - 5 - \cdots = i \cdot 2|E| - i^2$ haben muss.
Da beim Lollipopgraphen $|E| = \Theta(n^2)$ ist und Knoten u der Knoten $v_{n/2}$ ist, ergibt sich als Potenzialdifferenz zwischen u und v ein Wert der Größenordnung $\Theta(n^3)$.
Daraus ergibt sich natürlich auch $\mathcal{C}(L_n) = \Omega(n^3)$. Da wir schon $\mathcal{C}(G) < n^3$ für beliebige zusammenhängende Graphen G gezeigt hatten, ergibt sich insgesamt:

$$\mathcal{C}(L_n) = \Theta(n^3).$$

Die Schranke aus Satz 4.56 ist also für L_n asymptotisch optimal.

4.58 Beispiel. Andererseits ist Satz 4.56 unpräzise für den vollständigen Graphen K_n. Er liefert eine Schranke von $\mathcal{C}(K_n) = O(n^3)$, man kann aber auch eine Schranke von $\Theta(n \log n)$ zeigen. Dies wollen wir nun tun:

4.59 Behauptung. $\mathcal{C}(K_n) = \Theta(n \log n)$.

Beweis: (Skizze) Schreibe für einen konkreten Random Walk die Reihenfolge der besuchten Knoten auf.

1	4	5	1	3	5	7	...

Markiere die Knoten, die man zum ersten Mal sieht:

1	**4**	**5**	1	**3**	5	**7**	...

Sei X_i die Anzahl der Schritte zwischen der i-ten und der $(i+1)$-ten Markierung. In diesem Beispiel gilt $X_1 = 1, X_2 = 1, X_3 = 2$, usw.
Auf vollständigen Graphen wählt man beim Random Walk als Nachfolgerknoten einen Knoten aus $n-1$ Knoten. Daher gilt:

$$\textbf{Prob}\,(\text{es wird ein neuer Knoten besucht, wenn man schon } i \text{ Knoten gesehen hat}) = \frac{n-i}{n-1}.$$

Also ist $\mathbf{E}[X_i] = \frac{n-1}{n-i}$. Mit diesen Zufallsvariablen X_i können wir schreiben:

$$\begin{aligned}
\mathcal{C}(K_n) &= \mathbf{E}\,[\#\text{ Schritte, um alle Knoten zu sehen}] \\
&= \mathbf{E}\,[X_1 + X_2 + \cdots + X_{n-1}] \\
&= \sum_{i=1}^{n-1} \mathbf{E}\,[X_i] \\
&= \sum_{i=1}^{n-1} \frac{n-1}{n-i}.
\end{aligned}$$

Dies ist eine Formel, die wir schon einmal beim Coupon Collector Problem gesehen haben. Dort haben wir gesehen, dass gilt:

$$\sum_{i=1}^{n-1} \frac{n-1}{n-i} = (n-1)H_{n-1} = \Theta(n \log n).$$

□

Das Problem, alle Knoten zu sehen, ist überhaupt dem Coupon Collector Problem sehr ähnlich; der Unterschied besteht allerdings darin, dass man das gleiche Element nicht sofort noch einmal wählen darf.

Im Beispiel des vollständigen Graphen klafft eine große Lücke zwischen der Schranke $O(n^3)$ und dem tatsächlichen Wert $\Theta(n \log n)$. Es gibt jedoch auch bessere allgemeine Schranken. Um diese beschreiben zu können, benötigen wir folgende Definition:

4.60 Definition. $R(G) = \max_{u \neq v} R_{u,v}$ ist der „Widerstand des Graphen G“.

Wir betrachten zunächst ein Beispiel:

4.61 Beispiel. Der vollständige Graph K_n hat den Widerstand $R(K_n) = 2/n$.

Beweis: Wir betrachten ein beliebiges Knotenpaar $u \neq v$ aus K_n. Um R_{uv} zu bestimmen, müssen wir in u eine Einheit einspeisen und in v eine Einheit entnehmen. Man kann relativ leicht nachvollziehen, dass die folgenden Potenzialwerte eine zulässige Lösung ergeben: $pot_u = 2/n$, $pot_v = 0$ sowie $pot_x = 1/n$ für alle $x \notin \{u, v\}$. Daraus ergibt sich $R_{uv} = 2/n$. □

4.62 Satz. *Sei G ein zusammenhängender Graph mit m Kanten und Widerstand $R(G)$. Es gilt*

$$m \cdot R(G) \;\leq\; \mathcal{C}(G) \;\leq\; 2 \cdot e^3 \cdot m \cdot R(G) \cdot \ln n + n,$$

hierbei ist e die Eulersche Konstante.

4.63 Beispiel. Da $R(K_n) = 2/n$ für den vollständigen Graphen ist, wird die obere Schranke zu:

$$\mathcal{C}(K_n) \leq 2 \cdot e^3 \cdot \frac{n^2}{2} \cdot \frac{2}{n} \cdot \ln n + n = O(n \log n).$$

Hier ist Satz 4.62 also wesentlich besser als Satz 4.56.

Rechentechnischer Trick als Hilfe zum Beweis von Satz 4.62: Für alle natürlichen Zahlen $i \geq 0$ gilt:

$\mathcal{C}(G) \leq i + \mathbf{Prob}\,(\text{In einem Random Walk der Länge} \leq i \text{ werden nicht alle Knoten besucht}) \cdot \mathcal{C}(G)$

Beweis: (Skizze) Alle Random Walks der Länge $\leq i$, die alle Knoten besuchen, tragen zum Erwartungswert $\mathcal{C}(G)$ einen Wert $\leq i$ bei. Wenn man in den ersten i Schritten nicht alle Knoten gesehen hat, ist die erwartete Anzahl an Schritten, um danach die fehlenden Knoten zu sehen, $\leq \mathcal{C}(G)$. □

Beweis: (Satz 4.62, untere Schranke) Wähle u und v so, dass $R(G) = R_{u,v}$, also die zwei Knoten, die die Spannungsdifferenz maximieren. Da man in $R_{u,v}$ den Betrag der Spannungsdifferenz betrachtet, gilt $R_{u,v} = R_{v,u}$. Wir können daher u und v so wählen, dass o.B.d.A. $h_{u,v} \geq h_{v,u}$ gilt.
Nach Definition ist $\mathcal{C}(G) = \max_u \mathcal{C}_u(G)$. Andererseits ist $\mathcal{C}_u(G) \geq h_{u,v}$, da man bei einem erfolgreichen Durchlauf des Graphen ab dem Knoten u auch einen Pfad von u nach v durchlaufen hat. Wir brauchen also nur noch $h_{u,v}$ abzuschätzen, dies tun wir wie folgt:
Wir wissen:

a) $C_{u,v} = 2mR_{u,v}$. (Satz 4.53)

b) $C_{u,v} = h_{u,v} + h_{v,u}$. (Nach Definition.)

c) $C_{u,v} \leq 2h_{u,v}$. (Wegen b) und $h_{u,v} \geq h_{v,u}$).

Aus a) und c) folgt $h_{u,v} \geq m \cdot R_{u,v} = m \cdot R(G)$.

□

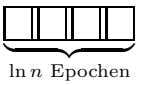
Beweis: (Satz 4.62, obere Schranke) Wähle einen beliebigen Knoten v fest. Betrachte den Random Walk der Länge $2 \cdot e^3 \cdot m \cdot R(G)$, den man in einem Knoten $x \neq v$ startet. Die erwartete Zeit, um von x nach v zu gelangen, ist

$$h_{x,v} \leq C_{x,v} = 2 \cdot m \cdot R_{x,v}(G) \leq 2 \cdot m \cdot R(G).$$

Hier betrachten wir einen Random Walk, der um den Faktor e^3 länger ist, und nach Markov ergibt sich

$$\textbf{Prob}\,(\text{Knoten } v \text{ wird dabei nicht besucht}) \leq \frac{1}{e^3}.$$

Betrachte nun einen Random Walk, der in x startet und der die Länge $2 \cdot e^3 \cdot m \cdot R(G) \cdot \ln n$ hat. Unterteile ihn in so genannte *Epochen*, und zwar in $\ln n$ Epochen der Länge $2 \cdot e^3 \cdot m \cdot R(G)$.



Damit v insgesamt nicht besucht wird, muss gelten, dass v in allen $\ln n$ Epochen nicht besucht wird.

$$\textbf{Prob}\,(\text{in einem solchen Random Walk wird } v \text{ nicht besucht}) \leq \left(\frac{1}{e^3}\right)^{\ln n} = \frac{1}{e^{3 \ln n}} = \frac{1}{n^3}.$$

$$\Rightarrow \textbf{Prob}\,(\text{es gibt einen Knoten, der in diesem Random Walk nicht besucht wird}) \leq n \cdot \frac{1}{n^3} = \frac{1}{n^2}.$$

Hier verwenden wir jetzt den Rechentechnischen Trick mit $i = 2 \cdot e^3 \cdot m \cdot R(G) \cdot \ln n$:

$$\mathcal{C}(G) \leq 2 \cdot e^3 \cdot m \cdot R(G) \cdot \ln n + \textbf{Prob}\,(\text{es werden nicht alle Knoten gesehen}) \cdot \mathcal{C}(G)$$

$$\mathcal{C}(G) \leq 2 \cdot e^3 \cdot m \cdot R(G) \cdot \ln n + \frac{1}{n^2} \cdot \mathcal{C}(G)$$

Mit der groben Abschätzung aus der letzten Vorlesung: $\mathcal{C}(G) \leq n^3$ folgt

$$\mathcal{C}(G) \leq 2 \cdot e^3 \cdot m \cdot R(G) \cdot \ln n + n.$$

□

Satz 4.62 liefert eine effiziente Methode, um $\mathcal{C}(G)$ abzuschätzen, $R(G)$ lässt sich durch Lösen von Gleichungssystemen ermitteln.

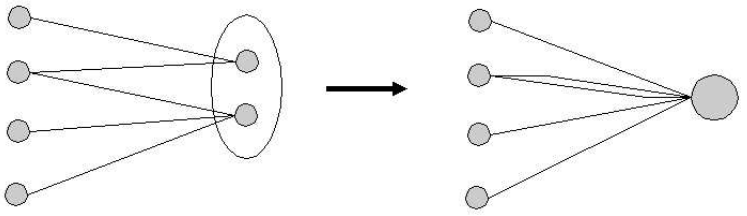
	Satz 4.56	Satz 4.62
vollständiger Graph	ungenau	genau
Lollipop-Graph	genau	ungenau (Faktor $\log n$)

Frage: Wenn man zusätzliche Kanten einfügt, verringert sich die Überdeckungszeit $\mathcal{C}(G)$?
Entgegen der Intuition lautet die Antwort: Nein.
Gegenbeispiel:

$$\mathcal{C}(\text{Kette}) = \Theta(n^2)$$
$$\mathcal{C}(L_n) = \Theta(n^3)$$

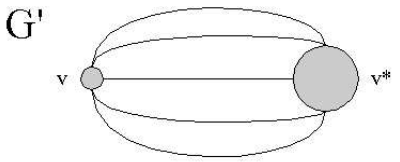
Der Lollipop-Graph entsteht aus einer Kette mit n Knoten durch Hinzufügen von Kanten zur $n/2$-Clique.
(Übrigens gilt ein ähnlich überraschendes Resultat auch bei der Verkehrsplanung: Manchmal lässt sich der Verkehrsfluss verbessern, indem man Straßen sperrt.)
Nun ein paar Fakten aus der Elektrotechnik, die wir hier nicht beweisen wollen, die sich aber auf Random Walks übertragen lassen:

Rayleighs Kurzschlussprinzip: Effektive Widerstände erhöhen sich nicht, wenn man zwei Knoten kurzschließt. Dabei werden zwei Knoten identifiziert, d.h. zu einem Knoten verschmolzen. Alle Kanten, die vorher zu einem der beiden Knoten inzident waren, sind nun inzident zum neuen Meta-Knoten.



4.64 Behauptung. Sei deg_{min} der minimale Grad in einem Graphen G. Dann folgt $R(G) \geq \frac{1}{deg_{min}}$.

Beweis: Wähle Knoten v mit minimalem Grad. Schließe alle $v' \in V \setminus \{v\}$ kurz. Der resultierende Graph G' besteht nur noch aus zwei Knoten v, v^* und deg_{min} Kanten zwischen v und dem Meta-Knoten v^*.



Da die Kanten parallel geschaltet sind, gilt

$$R_{v,v^*}(G') = \frac{1}{deg_{min}} \qquad \Rightarrow \qquad R(G) \geq \frac{1}{deg_{min}}.$$

□

4.7 Algorithmen für 2-SAT und 3-SAT

Wir stellen einen Algorithmus für 2-SAT vor, der eine polynomielle worst-case-erwartete Laufzeit besitzt. Darüber hinaus stellen wir zwei Algorithmen für 3-SAT mit erwarteter Laufzeit

$O(1.3334^n)$ bzw. $O(1.3302^n)$ vor. Zunächst zum 2-SAT-Algorithmus:

Algorithmus A: Initialisiere die Belegung $a := (0, 0, \ldots, 0)$.
FOR $t = 1$ TO ∞ DO {
    IF a erfüllend THEN STOP
    ELSE {
        Wähle eine Klausel C, die unter der Belegung a nicht erfüllt ist, o.B.d.A. sei $C = x_1 \vee x_2$.
        Wähle $i \in \{1, 2\}$ so, dass $\mathbf{Prob}\,(i = 1) = 1/2$ ist, und flippe Bit a_i.
    }
}

4.65 Behauptung. Gegeben sei eine 2-SAT-Instanz. Wenn eine erfüllende Belegung existiert, dann findet Algorithmus A in erwarteter Schrittzahl $O(n^2)$ eine erfüllende Belegung.

Beweis: Bezeichne mit $H(a, b) := \mid \{i \mid a_i \neq b_i\} \mid$ die Hammingdistanz zwischen zwei Vektoren a und b. Wähle eine erfüllende Belegung a^* (diese ist im allgemeinen nicht eindeutig) und betrachte den Verlauf von $H(a, a^*)$ als Random Walk.
Wenn ein Bit geflippt wird, so vergrößert sich die Hammingdistanz zwischen a und a^* entweder um 1, oder sie verringert sich um 1. Nach Wahl der Klausel im Algorithmus ist $a_1 = a_2 = 0$. Wir betrachten die möglichen Fälle: Wenn $a_1^* = 1$ und $a_2^* = 0$ ist, dann verringert sich $H(a, a^*)$ mit Wahrscheinlichkeit $1/2$, ebenso, wenn $a_1^* = 0$ und $a_2^* = 1$ ist. Wenn $a_1^* = a_2^* = 1$ ist, dann verringert sich die Hammingdistanz sogar mit Wahrscheinlichkeit 1. Wir erinnern uns an den Random Walk, den wir zu Beginn von Abschnitt 4.6 analysiert haben und erhalten, dass die erwartete Anzahl an Schritten, um zu Position 0 zu gelangen, höchstens n^2 ist. □

Wenn wir *nicht* voraussetzen, dass es eine erfüllende Belegung gibt, sondern entscheiden wollen, *ob* es eine erfüllende Belegung gibt, können wir den Algorithmus folgendermaßen modifizieren:

Algorithmus B: Ersetze „∞“ in Algorithmus A durch „$2n^2$“ und nach Ablauf der FOR-Schleife gib aus „es gibt (vermutlich) keine erfüllende Belegung.“

Angenommen, es existiert eine erfüllende Belegung, dann sei X die Zufallsvariable, die die Anzahl der Schritte angibt, bis eine erfüllende Belegung gefunden wird. Es gilt nach Markov $\mathbf{Prob}\,(X \geq 2 \cdot \mathbf{E}\,[X]) \leq 1/2$. Also ist die Wahrscheinlichkeit, dass Algorithmus B eine erfüllende Belegung findet, mindestens $1/2$. Da Algorithmus B einen einseitigen Fehler hat, kann dieser mit den üblichen Methoden verringert werden.
Nun kommen wir zu einem 3-SAT-Algorithmus aus dem Jahr 1999 mit einer erwarteten Laufzeit von $O(1.3334^n)$. Zuerst definieren wir den Begriff der Entropie: Für $0 \leq \alpha \leq 1$ sei

$$H(\alpha) := -\alpha \cdot \log \alpha - (1 - \alpha) \cdot \log(1 - \alpha).$$

Anmerkungen:

a) Eigentlich ist $\log 0$ nicht definiert, aber aus Stetigkeitsgründen definiert man $H(0) = H(1) = 0$.

b) Es gilt $2^{-H(\alpha)} = \alpha^\alpha \cdot (1 - \alpha)^{1-\alpha}$.

Wir werden die Entropie benutzen, um Binomialkoeffizienten geeignet abzuschätzen. Insbesondere verwenden wir die Abschätzung aus dem folgenden Satz, wobei wir bemerken wollen, dass es sogar genauere Abschätzungen gibt.

4.66 Satz. *Für alle* $0 \leq k \leq n$ *gilt* $\frac{1}{n+1} \cdot 2^{H(k/n) \cdot n} \leq \binom{n}{k} \leq 2^{H(k/n) \cdot n}$.

Beweis: Die Fälle $k = 0$ und $k = n$ sind trivial. Sei nun $k \in \{1, \ldots, n-1\}$ und $p := k/n$. Wir werfen eine Münze, die mit Wahrscheinlichkeit p eine 1 liefert. $f(i)$ sei die Wahrscheinlichkeit, bei n solchen Münzwürfen genau i Einsen zu bekommen, d.h. $f(i) = \binom{n}{i}\ p^i(1-p)^{n-i}$ und $\sum_{i=0}^{n} f(i) = 1$. Wir zeigen nun, dass $f(k)$ maximal ist, also $f(k) = \max_{0 \le i \le n} f(i)$ ist. Wir weisen dies nach, indem wir uns die Quotienten aus „benachbarten" $f(i)$-Werten anschauen:

$$\frac{f(i+1)}{f(i)} = \frac{\binom{n}{i+1} \cdot p^{i+1} \cdot (1-p)^{n-(i+1)}}{\binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}} = \frac{p}{1-p} \cdot \frac{n-i}{i+1}.$$

Dieser Quotient ist eine monoton in i fallende Funktion. Weiterhin gilt:

$$\begin{aligned} \frac{f(k+1)}{f(k)} &= \frac{p}{1-p} \cdot \frac{n-k}{k+1} = \frac{k/n}{(n-k)/n} \cdot \frac{n-k}{k+1} = \frac{k}{k+1} < 1 \text{ sowie} \\ \frac{f(k)}{f(k-1)} &= \frac{k}{n-k} \cdot \frac{n-k+1}{k} = \frac{n-k+1}{n-k} > 1. \end{aligned}$$

Also ist $f(k)$ maximal.

Aus $\sum_{i=0}^{n} f(i) = 1$ und der Maximalität von $f(k)$ folgt $\sum_{i=0}^{n} f(k) \ge 1$. Also ist $f(k) \ge \frac{1}{n+1}$ und $f(k) \le 1$. Wir stellen $f(k)$ anders dar:

$$\begin{aligned} f(k) &= \binom{n}{k} \left(\frac{k}{n}\right)^k \left(\frac{n-k}{n}\right)^{n-k} \\ &= \binom{n}{k} \left[\left(\frac{k}{n}\right)^{k/n} \left(1 - \frac{k}{n}\right)^{1-k/n}\right]^n \\ &= \binom{n}{k} \left[2^{-H(k/n)}\right]^n. \end{aligned}$$

Aus $1/(n+1) \le f(k) \le 1$ folgt nun:

$$\binom{n}{k} \ge \frac{1}{n+1} \cdot 2^{H(k/n) \cdot n} \text{ und } \binom{n}{k} \le 2^{H(k/n)n}.$$

□

Wir stellen nun den 3-SAT-Algorithmus RandomWalk vor. Sei a eine initiale Variablenbelegung.

Algorithmus `RandomWalk(`a`)`
DO $3n + 1$ times {
    IF a erfüllend THEN STOP
    ELSE {
        Wähle eine Klausel C, die unter der Belegung a
        nicht erfüllt ist, o.B.d.A. sei $C = x_1 \vee x_2 \vee x_3$.
        Wähle $i \in \{1, 2, 3\}$ gemäß der Gleichverteilung und flippe a_i.
    }
}
output: Es gibt (vermutlich) keine erfüllende Belegung.

Wir beweisen zunächst folgende Aussage:

4.67 Satz. *Sei $a \in \{0,1\}^n$ und a^* eine erfüllende Belegung. Dann gilt:*

$$\textbf{Prob}\,(\textit{RandomWalk(a) findet eine erfüllende Belegung}) \geq \frac{1}{3n+1}\left(\frac{1}{2}\right)^{d(a,a^*)},$$

wenn $d(v,w)$ die Hammingdistanz zweier Vektoren v und w bezeichnet.

Beweis: Wir betrachten den Verlauf der Hammingdistanz $d(a,a^*)$. Mit der gleichen Begründung wie bei der Analyse des 2-SAT Algorithmus verringert sich diese in jedem Schritt mit einer Wahrscheinlichkeit von mindestens 1/3. Mit der Sichtweise „Hammingdistanz um 1 verringern" entspricht „Schritt nach links" und „Hammingdistanz um 1 vergrößern" entspricht „Schritt nach rechts" erhalten wir, wenn wir auch $j := d(a,a^*)$ abkürzen:

$$\begin{array}{c}\textbf{Prob}\,(\text{in } 3n \text{ Schritten wird eine erfüllende Belegung gefunden})\\ \geq \textbf{Prob}\,(\text{in } 3j \text{ Schritten wird eine erfüllende Belegung gefunden})\\ \geq\\ \textbf{Prob}\,(\text{bei } 3j \text{ Schritten werden mind. } 2j \text{ Schritte nach links und höchstens } j \text{ nach rechts gemacht})\end{array}$$

$$\begin{aligned}
&\geq \binom{3j}{2j}\left(\frac{1}{3}\right)^{2j}\left(\frac{2}{3}\right)^{j}\\
&\geq \frac{1}{3j+1}2^{H(2/3)\cdot 3j}\left(\frac{1}{3}\right)^{2j}\left(\frac{2}{3}\right)^{j}\\
&= \frac{1}{3j+1}2^{H(2/3)\cdot 3j}\left[\left(\frac{1}{3}\right)^{1/3}\left(\frac{2}{3}\right)^{2/3}\right]^{3j}\left(\frac{1}{2}\right)^{j}\\
&= \frac{1}{3j+1}2^{H(2/3)\cdot 3j}\left[2^{-H(2/3)}\right]^{3j}\left(\frac{1}{2}\right)^{j}\\
&= \frac{1}{3j+1}\left(\frac{1}{2}\right)^{j}\\
&\geq \frac{1}{3n+1}\left(\frac{1}{2}\right)^{j}
\end{aligned}$$

□

Man beachte übrigens, dass die erfüllende Belegung, die der Algorithmus in der Aussage des Satzes findet, nicht unbedingt die Belegung a^* sein muss.
Als Korollar erhalten wir übrigens aus dem Satz, dass der Algorithmus „RandomWalk($00\cdots0$); RandomWalk($11\cdots1$);" mit Wahrscheinlichkeit mindestens

$$\frac{1}{3n+1}\cdot\left(\frac{1}{2}\right)^{n/2} = \Omega(1.42^{-n})$$

eine erfüllende Belegung findet, da a^* entweder zum Einsvektor $11\cdots1$ oder zum Nullvektor $00\cdots0$ einen durch $n/2$ beschränkten Hammingabstand hat. Mit dem üblichen Argument bekäme man einen 3SAT-Algorithmus mit erwarteter Laufzeit $O(1.42^n)$. Wir gehen aber „intelligenter" vor und würfeln das initiale a nach der Gleichverteilung aus.
Der gesamte Algorithmus RW sieht wie folgt aus:
Algorithmus RW: Wähle gemäß Gleichverteilung $a \in \{0,1\}^n$ und starte RandomWalk(a).

Zur Analyse von RW definiere zunächst die Zufallsvariable $X_i := d(a_i, a_i^*)$. Diese ist 1, wenn sich (das gewürfelte) a und a^* in der i-ten Stelle unterscheiden und 0 sonst. Man kann nun

$$d(a,a^*) = X_1 + \cdots + X_n$$

schreiben. Wie groß ist die Erfolgswahrscheinlichkeit P_{Erfolg}, dass Algorithmus RW eine erfüllende Belegung findet? Es ist

$$\begin{aligned} P_{\text{Erfolg}} &\geq \sum_{a\in\{0,1\}^n} \mathbf{Prob}\,(a \text{ gewürfelt}) \cdot \frac{1}{3n+1} \cdot \left(\frac{1}{2}\right)^{H(a,a^*)} \\ &= \frac{1}{3n+1} \cdot \sum_{a\in\{0,1\}^n} \mathbf{Prob}\,(a \text{ gewürfelt}) \cdot \left(\frac{1}{2}\right)^{d(a,a^*)} \\ &= \frac{1}{3n+1} \cdot \mathbf{E}\left[\left(\frac{1}{2}\right)^{d(a,a^*)}\right] \\ &= \frac{1}{3n+1} \cdot \mathbf{E}\left[\left(\frac{1}{2}\right)^{X_1+\cdots+X_n}\right]. \end{aligned}$$

Schließlich gilt $\mathbf{E}\left[\left(\frac{1}{2}\right)^{X_1+\cdots+X_n}\right] = \mathbf{E}\left[\left(\frac{1}{2}\right)^{X_1}\right]\cdots\mathbf{E}\left[\left(\frac{1}{2}\right)^{X_n}\right]$, da $\mathbf{E}\,[X\cdot Y] = \mathbf{E}\,[X]\cdot\mathbf{E}\,[Y]$ für unabhängige Zufallsvariablen X und Y gilt. Hier sind die X_i und somit die $(\frac{1}{2})^{X_i}$ unabhängig.

Da $X_i \in \{0,1\}$, ist $\mathbf{E}\left[\left(\frac{1}{2}\right)^{X_i}\right] = (1/2)\cdot\left(\frac{1}{2}\right)^1 + (1/2)\cdot\left(\frac{1}{2}\right)^0 = 3/4$.
(Man beachte übrigens wieder einmal, dass *nicht* $\mathbf{E}\left[\left(\frac{1}{2}\right)^{X_i}\right] = (1/2)^{\mathbf{E}[X_i]}$ ist!)
Insgesamt haben wir erhalten:

$$P_{\text{Erfolg}} \geq \frac{1}{3n+1}\cdot(3/4)^n.$$

Mit dem üblichen Argument erhalten wir also einen randomisierten 3-SAT-Algorithmus mit erwarteter Laufzeit $(4/3)^n \cdot poly(n)$.
Nun kommen wir zu einer Verbesserung: Bei der Initialisierung ignorieren wir ja bislang die Klauseln völlig. Falls beispielsweise $C = x_1 \vee x_2 \vee x_3$ ist, dann werden mit Wahrscheinlichkeit $1/8$ alle drei Variablen auf Null gesetzt. Kann man das nicht vermeiden?

Die prinzipielle Idee besteht darin, die Variablen in Dreiergruppen (entsprechend den Klauseln) auszuwürfeln und dabei zu vermeiden, alle drei gleichzeitig auf Null zu setzen. Das Problem dabei ist aber, dass Klauseln Variablen gemeinsam haben können und es daher Überlappungen zwischen den Dreiergruppen gibt. Wir können aber versuchen, möglichst viele unabhängige Dreiergruppen zu finden. Dies führt zu folgender Definition:

4.68 Definition. Zwei Klauseln heißen *unabhängig*, wenn sie keine Variablen gemeinsam haben.

Beispiel: $x_1 \vee x_2 \vee \overline{x}_5$ und $x_3 \vee x_4 \vee x_5$ sind *nicht* unabhängig.

Wir starten unseren neuen Algorithmus zunächst damit, dass wir eine (inklusions-)maximale Menge unabhängiger Klauseln berechnen. Wenn wir eine Menge M von Klauseln als Eingabe für das 3-SAT-Problem gegeben haben, dann heißt eine Teilmenge $T \subseteq M$ der Klauseln maximal unabhängig, wenn je zwei von ihnen unabhängig sind und wir keine Klausel aus $M \setminus T$ zu T hinzufügen können, ohne diese Eigenschaft zu verletzen. Es sollte klar sein, dass man eine solche maximale Menge von unabhängigen Klauseln greedy berechnen kann. Sei im folgenden $\widehat{m}$ die Anzahl der unabhängigen Klauseln, die man gefunden hat. Nach geeigneter Umbenennung kann man davon ausgehen, dass diese $\widehat{m}$ Klauseln von der Form sind

$$\begin{aligned} C_1 &= x_1 \vee x_2 \vee x_3 \\ C_2 &= x_4 \vee x_5 \vee x_6 \\ &\vdots \\ C_{\widehat{m}} &= x_{3\widehat{m}-2} \vee x_{3\widehat{m}-1} \vee x_{3\widehat{m}}. \end{aligned}$$

Wir initialisieren die Variablen in den unabhängigen Klauseln (also $x_1, \ldots, x_{3\widehat{m}}$) anders als vorhin, mit Hilfe einer parametrisierten Prozedur namens Ind-Clauses-Assign. Die Parameter p_1, p_2, p_3 müssen dabei die Gleichung $3p_1 + 3p_2 + p_3 = 1$ erfüllen.

Ind-Clauses-Assign(p_1, p_2, p_3)
Für jede Klausel $C = x_i \vee x_j \vee x_k \in \{C_1, \ldots, C_{\widehat{m}}\}$ **do:**
Setze die drei Variablen mit Wahrscheinlichkeit:

p_1 auf $(0,0,1)$	p_2 auf $(0,1,1)$	p_3 auf $(1,1,1)$
p_1 auf $(0,1,0)$	p_2 auf $(1,1,0)$	
p_1 auf $(1,0,0)$	p_2 auf $(1,0,1)$	

Unser neuer Algorithmus, nennen wir ihn RW*, sieht wie folgt aus, die Parameter p_1, p_2, p_3 wählen wir dabei erst später:

Algorithmus RW$^*(p_1, p_2, p_3)$
Ind-Clauses-Assign(p_1, p_2, p_3)
Initialisiere die übrigen Variablen, also $x_{3\widehat{m}+1}, \ldots, x_n$, jeweils auf 0 bzw. 1, jeweils mit Wahrscheinlichkeit 1/2.
Nun sind alle Variablen gesetzt, wir haben also eine Belegung a erhalten.
Starte RandomWalk(a).

Welche Erfolgswahrscheinlichkeit hat RW$^*(p_1, p_2, p_3)$? Wieder müssen wir $\mathbf{E}\left[\left(\frac{1}{2}\right)^{d(a,a^*)}\right]$ analysieren. Aber diesmal sind nicht alle Bits unabhängig gewählt. Daher definiere:

$$\begin{aligned} X_{1,2,3} &:= d(\ (a_1, a_2, a_3), (a_1^*, a_2^*, a_3^*)\ ), \\ X_{4,5,6} &:= d(\ (a_4, a_5, a_6), (a_4^*, a_5^*, a_6^*)\ ), \text{ etc.} \end{aligned}$$

Damit ist dann

$$d(a, a^*) = X_{1,2,3} + X_{4,5,6} + \cdots + X_{3\widehat{m}-2,3\widehat{m}-1,3\widehat{m}} + X_{3\widehat{m}+1} + X_{3\widehat{m}+2} + \cdots + X_n$$

und es ergibt sich:

$$P_{\text{Erfolg}} \geq \frac{1}{3n+1} \cdot \mathbf{E}\left[(\tfrac{1}{2})^{X_{1,2,3}}\right] \cdot \mathbf{E}\left[(\tfrac{1}{2})^{X_{4,5,6}}\right] \cdots \mathbf{E}\left[(\tfrac{1}{2})^{X_{3\widehat{m}-2,3\widehat{m}-1,3\widehat{m}}}\right] \cdot \prod_{i=3\widehat{m}+1}^{n} \mathbf{E}\left[(\tfrac{1}{2})^{X_i}\right].$$

Das hintere Produkt $\prod_{i=3\widehat{m}+1}^{n} \mathbf{E}\left[(\frac{1}{2})^{X_i}\right]$ können wir genauso analysieren, wie wir das vorhin getan haben, es ergibt sich, dass sein Wert $(3/4)^{n-3\widehat{m}}$ ist.
Bleibt die Aufgabe, $\mathbf{E}\left[(\frac{1}{2})^{X_{1,2,3}}\right]$ zu analysieren. (Die anderen Terme ergeben sich analog.) Dieser Erwartungswert hängt von a_1^*, a_2^*, a_3^* ab. Wir müssen drei Fälle betrachten, je nachdem, wieviele Einsen die drei a_1^*, a_2^*, a_3^* enthalten:

Fall $a_1^* + a_2^* + a_3^* = 3$.
Wir müssen alle sieben Möglichkeiten für die Belegung der a_1, a_2, a_3 durchgehen. Wenn wir z.B. $(a_1, a_2, a_3) = (0, 0, 1)$ wählen (was mit Wahrscheinlichkeit p_1 passiert), dann haben wir $X_{1,2,3} = 2$. Wenn wir $(a_1, a_2, a_3) = (0, 1, 1)$ wählen (was mit Wahrscheinlichkeit p_2 passiert), dann haben wir $X_{1,2,3} = 1$.
Gehen wir alle sieben Möglichkeiten durch, so erhalten wir:

$$\begin{aligned}\mathbf{E}\left[(\frac{1}{2})^{X_{1,2,3}}\right] &= (\frac{1}{2})^0 \cdot p_3 + (\frac{1}{2})^1 \cdot 3p_2 + (\frac{1}{2})^2 \cdot 3p_1 \\ &= p_3 + \frac{3}{2}p_2 + \frac{3}{4}p_1.\end{aligned}$$

Nun betrachten wir noch die anderen beiden Fälle:

Fall $a_1^* + a_2^* + a_3^* = 1$:

$$\mathbf{E}\left[(\frac{1}{2})^{X_{1,2,3}}\right] = (\frac{1}{2})^0 \cdot p_1 + (\frac{1}{2})^1 \cdot 2p_2 + (\frac{1}{2})^2 \cdot (2p_1 + p_3) + (\frac{1}{2})^3 \cdot p_2 = \frac{p_3}{4} + \frac{9}{8} \cdot p_2 + \frac{3}{2} \cdot p_1.$$

Fall $a_1^* + a_2^* + a_3^* = 2$:

$$\mathbf{E}\left[(\frac{1}{2})^{X_{1,2,3}}\right] = (\frac{1}{2})^0 \cdot p_2 + (\frac{1}{2})^1 \cdot (p_3 + 2p_1) + (\frac{1}{2})^2 \cdot 2p_2 + (\frac{1}{2})^3 \cdot p_1 = \frac{p_3}{2} + \frac{3}{2} \cdot p_2 + \frac{9}{8} \cdot p_1.$$

Wählen wir nun $p_1 := 4/21, p_2 := 2/21, p_3 := 3/21$, dann stellt sich heraus, dass der Erwartungswert $\mathbf{E}\left[(\frac{1}{2})^{X_{1,2,3}}\right]$ in allen drei Fällen gleich ist, nämlich $3/7$. Damit ergibt sich als Erfolgswahrscheinlichkeit für $\mathrm{RW}^*(4/21, 2/21, 3/21)$:

$$\frac{1}{p(n)} \cdot \left(\frac{3}{4}\right)^{n-3\widehat{m}} \cdot \left(\frac{3}{7}\right)^{\widehat{m}} = \left(\frac{3}{4}\right)^{n} \cdot \left(\frac{64}{63}\right)^{\widehat{m}} \cdot \frac{1}{p(n)}.$$

Diese Erfolgswahrscheinlichkeit wächst mit $\widehat{m}$, ist aber nur bei großen $\widehat{m}$ asymptotisch besser als beim alten Algorithmus!
Der Ausweg: Wir brauchen einen Algorithmus, der bei 3SAT-Instanzen mit *kleinem* $\widehat{m}$ erfolgreich ist. Den gibt es aber: Wenn man die Variablen $x_1, \ldots, x_{3\widehat{m}}$ belegt hat, dann bleibt nur noch eine 2SAT-Instanz übrig: Da die Klauselmenge *maximal* unabhängig war, enthält jede Klausel in der Eingabeinstanz eine der Variablen $x_1, \ldots, x_{3\widehat{m}}$. 2SAT kann jedoch in polynomieller Zeit gelöst werden. Dies führt zu folgendem Algorithmus RED2:

RED2 (q_1, q_2, q_3)
Ind-Clauses-Assign(q_1, q_2, q_3)
Löse 2SAT auf den restlichen Klauseln.

RED2$(1/7, 1/7, 1/7)$ ist z.B. dann erfolgreich, wenn die Belegungen $a_1, \ldots, a_{3\widehat{m}}$ alle mit $a_1^*, \ldots, a_{3\widehat{m}}^*$ übereinstimmen. Dies passiert mit Wahrscheinlichkeit mindestens $(1/7)^{\widehat{m}}$. Wir kombinieren beide Algorithmen:

MIX
RW$(\frac{4}{21}, \frac{2}{21}, \frac{3}{21})$.
RED2$(\frac{1}{7}, \frac{1}{7}, \frac{1}{7})$.

Nun ist die Erfolgswahrscheinlichkeit mindestens $\frac{1}{3n+1} \cdot \max\left[(\frac{1}{7})^{\widehat{m}}, \quad \left(\frac{3}{4}\right)^{n-3\widehat{m}} \cdot \left(\frac{3}{7}\right)^{\widehat{m}}\right]$.
Da die eine Funktion monoton wachsend in $\widehat{m}$ ist und die andere monoton fallend in $\widehat{m}$, erhalten wir eine untere Schranke, wenn wir das $\widehat{m}$ berechnen, wo beide den gleichen Wert ergeben: Es ergibt sich Gleichheit für $\widehat{m} \approx 0.1466525 \cdot n$. Dieses $\widehat{m}$ eingesetzt zeigt, dass die Erfolgswahrscheinlichkeit mindestens $P_{\text{Erfolg}} \geq 1.330258^{-n}$ ist.
Eine weitere Verbesserung auf die Erfolgswahrscheinlichkeit $P_{\text{Erfolg}} \geq 1.330193^{-n}$ kann man bekommen, wenn man die Wahl der Parameter p_1, p_2, p_3 noch optimiert, darauf wollen wir aber in der Vorlesung nicht eingehen.
Wir halten fest (vgl. unsere Vorüberlegungen vom Beginn dieses Abschnitts):

4.69 Satz. *Es gibt einen randomisierten Algorithmus, der das 3SAT-Problem in worst-case-Laufzeit $O(1.3302^n)$ löst. Genauer: auf Instanzen, die nicht erfüllbar sind, gibt er immer „nicht erfüllbar“ aus, und auf Instanzen, die erfüllbar sind, gibt er mit Wahrscheinlichkeit mindestens $1 - 2^{-n}$ eine erfüllende Belegung aus.*

Die Wahrscheinlichkeit $1 - 2^{-n}$ erhält man dabei mit den üblichen Amplifikationsargumenten.

4.8 Random Walks und Graphzusammenhang

Problem USTCON („Undirected s-t-connectivity“)
Gegeben: Ein ungerichteter Graph G sowie zwei Knoten s und t.
Aufgabe: Entscheide, ob es einen Weg zwischen s und t gibt.

*Laufzeit*effiziente Verfahren sind bekannt, so zum Beispiel DFS, das dieses Problem in Linearzeit löst. Allerdings benötigt DFS linearen Extraplatz (da die Knoten entlang eines Weges abgespeichert werden bzw. für jeden Knoten festgehalten wird, ob er schon besucht wurde oder nicht).
Ein *nichtdeterministischer* Algorithmus könnte wie folgt vorgehen: Rate den Weg von s nach t und verifiziere, dass es in der Tat ein Weg ist. Für die Verifikation muss sich der Algorithmus jeweils merken, an welchem Knoten er sich befindet, und kommt dabei mit $O(\log n)$ Extraplatz aus. (Das ist der Platz, den man zum Speichern einer Knotennummer benötigt.)
Der Satz von Savitch (den man eventuell aus der Vorlesung Komplexitätstheorie kennt) gibt es einen *deterministischen* Algorithmus, der mit $O(\log^2 n)$ Platz auskommt. (Die Laufzeit interessiert uns hierbei allerdings nicht.) Übrigens weiß man seit Ende des Jahres 2004 aus einer Arbeit von Omer Reingold, dass das Problem USTCON auch deterministisch auf logarithmischem Platz gelöst werden kann:
Siehe z.B. `http://www.eccc.uni-trier.de/eccc-reports/2004/TR04-094/index.html`
Wir kümmern uns hier um die randomisierten Algorithmen.

4.70 Definition (RLP — randomized logarithmic space and polynomial time)**.** Die Klasse RLP enthält alle Sprachen L, für die es eine probabilistische Turingmaschine gibt, die nur logarithmisch viel Extraplatz benötigt, die polynomielle Laufzeit hat und für die gilt:

$$\mathbf{Prob}\,(M \text{ akzeptiert } x) \begin{cases} \geq 1/2, & \text{falls } x \in L \\ = 0, & \text{falls } x \notin L \end{cases}.$$

Für den Begriff „Extraplatz“ nimmt man an, dass die Maschine ihr Eingabeband nur lesen darf, und Schreiboperationen nur auf einem zweiten Arbeitsband ausgeführt werden können, dessen Platzbedarf gezählt wird.

Unsere ganzen Vorüberlegungen zu den Random Walks in Graphen können nun benutzt werden, um folgendes zu zeigen:

4.71 Satz. *USTCON* $\in$ **RLP**.

Beweis: Die probabilistische Maschine M arbeitet wie folgt: Simuliere einen Random Walk der Länge $2n^3$ auf dem Graphen, beginnend im Knoten s. Falls in diesem Random Walk der Knoten t besucht wurde, gib „JA“ aus, sonst „NEIN“.
Zunächst beobachten wir, dass die Maschine auf jeden Fall „NEIN“ ausgibt, wenn es keinen Weg von s nach t gibt. Wenn es jedoch einen Weg von s nach t gibt, dann liegen s und t in einer Zusammenhangskomponente, die Anzahl Knoten dieser möge N heißen. Auf die Zusammenhangskomponente kann man unsere Sätze die Random Walks betreffend anwenden.
Nach Korollar 4.55 (S. 102) wissen wir, dass $C_{s,t} < N^3 \leq n^3$ ist. Nach Markov ist also die Wahrscheinlichkeit, dass in $2n^3$ Schritten der Knoten t nicht angetroffen wird, kleiner gleich $1/2$.
Was wird auf dem Arbeitsband verwaltet? Der Zähler, der bis $2n^3$ hochzählt, sowie die Nummer des Knotens, an dem man sich befindet, etc. Man kann sich überlegen, dass man dann insgesamt

mit Extraplatz $O(\log n)$ auskommt. □

Dies ist ein uniformer randomisierter **RLP**-Algorithmus für USTCON, der mit Polynomialzeit und logarithmisch viel Extraplatz auskommt. Man kann ihn (auf nicht-uniforme) Art und Weise deterministisch machen, und zwar mit *universellen Durchlaufsequenzen.*

4.8.1 Universelle Durchlaufsequenzen

In diesem Unterabschnitt beschränken wir uns der notationellen Einfachheit halber auf d-reguläre Graphen, also solche, in denen alle Knoten Grad d haben.
Ein Graph heiße gelabelt, wenn an jedem Knoten die zu ihm inzidenten d Kanten von 1 bis d durchnumeriert sind. Abbildung 4.4 zeigt ein Beispiel eines gelabelten Graphen. Man kann einen Graphen zum Beispiel labeln, indem man jeder Kante die Position in der entsprechenden Adjazenzliste zuweist.

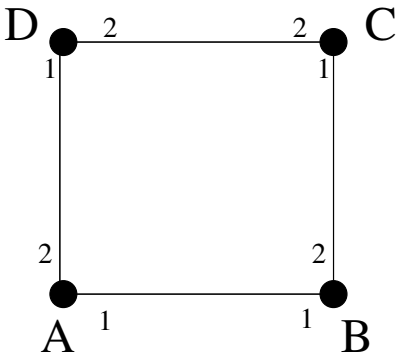


Abbildung 4.4: Ein *gelabelter* Graph.

Wir beobachten: Eine Folge $\sigma \in \{1, \ldots, d\}^r$ legt einen Durchlauf der Länge r durch den Graphen fest, wenn wir einen Startknoten vorgeben.
Beispiel: Wenn der Startknoten A ist, durchlaufen wir bei der Folge $1, 1, 1$ die Knotenfolge A, B, A, B. Wenn der Startknoten C ist, durchlaufen wir bei der Folge $1, 1, 1$ die Knotenfolge C, B, A, B.

4.72 Definition. Eine Folge σ *durchläuft* einen gelabelten Graphen G *vollständig*, wenn unabhängig vom Startknoten ein Durchlauf durch G gemäß σ alle Knoten von G mindestens einmal besucht.

Zum Beispiel durchläuft die Folge $2, 1, 1, 2$ den oben gezeigten Graphen vollständig.
Sei $\mathcal{G}$ eine endliche Menge von zusammenhängenden d-regulären gelabelten Graphen auf n Knoten. (Achtung: „Gleiche“ Graphen mit verschiedenen Labellings zählen als verschiedene Elemente in $\mathcal{G}$.)

4.73 Definition. Eine Folge σ heißt *universelle Durchlaufsequenz* für eine solche Klasse $\mathcal{G}$, wenn σ jeden gelabelten Graphen G aus $\mathcal{G}$ vollständig durchläuft.

Eine universelle Durchlaufsequenz kann natürlich von einem deterministischen Algorithmus benutzt werden, um das Problem USTCON zu lösen, wenn der gelabelte Eingabegraph aus der Klasse $\mathcal{G}$ ist. Dies kann einfach durch Durchlaufen des Eingabegraphen gemäß σ geschehen. Leider jedoch gibt es keine guten Algorithmen, um solche universellen Durchlaufsequenzen zu berechnen. (Dies müsste ja in logarithmischem Platz gemacht werden.)
Man kann jedoch probabilistisch die Existenz von kurzen universellen Durchlaufsequenzen nachweisen. Dies wollen wir im folgenden tun.

4.74 Definition. Für eine nicht-leere Menge $\mathcal{G}$ von gelabelten, zusammenhängenden, d-regulären Graphen bezeichne $U(\mathcal{G})$ die Länge einer kürzesten universellen Durchlaufsequenz für $\mathcal{G}$.

Für die Aussage der folgenden Behauptung benötigen wir den Widerstand einer Graphklasse: $R(\mathcal{G})$ bezeichne den größten Widerstand eines Graphen aus $\mathcal{G}$.

4.75 Behauptung. $U(\mathcal{G}) \leq 5 \cdot |E| \cdot R(\mathcal{G}) \cdot \log(n^2 \cdot |\mathcal{G}|)$.

Beweis: Ein Random Walk durch einen d-regulären Graphen auf n Knoten entspricht einer Folge von Zahlen aus $\{1, \ldots, d\}$. Wir würfeln eine solche Folge aus, die die Länge $5 \cdot |E| \cdot R(\mathcal{G}) \cdot \log(n^2 \cdot |\mathcal{G}|)$ hat und analysieren die Wahrscheinlichkeit, dass auf einem Graphen G aus $\mathcal{G}$ ein bestimmter Knoten v nicht besucht worden ist.
Wie beim Beweis von Satz 4.62 unterteilen wir den Random Walk in $\log(n^2 \cdot |\mathcal{G}|)$ kleine Random Walks (auch „Epochen“ genannt) der Länge $5 \cdot |E| \cdot R(\mathcal{G})$.
Wir untersuchen für fest gewählte G, u und v, wie groß die Wahrscheinlichkeit ist, dass bei Start in u der Knoten v nicht besucht wird.
Wenn wir die Epoche in einem Knoten u' starten, dann ist nach Satz 4.53 die erwartete Anzahl an Schritten, bis man v zum ersten Mal sieht, durch

$$C_{u',v} \leq 2 \cdot |E| \cdot R(G) \leq 2|E| \cdot R(\mathcal{G})$$

beschränkt. Nach Markov ist also die Wahrscheinlichkeit, in einer Epoche der Länge $5 \cdot m \cdot R(\mathcal{G})$ den Knoten v nicht anzutreffen, durch $2/5$ beschränkt.

Die Wahrscheinlichkeit, dass v auf der gesamten Irrfahrt nicht angetroffen wird, ist damit für ein $c > 1$ beschränkt durch

$$\left(\frac{2}{5}\right)^{\log(n^2 \cdot |\mathcal{G}|)} = (n^2 \cdot |\mathcal{G}|)^{\log(2/5)} = \frac{1}{(n^2 \cdot |\mathcal{G}|)^c}.$$

Die Wahrscheinlichkeit, dass es irgendeinen Graphen G aus $\mathcal{G}$, einen Startknoten u und einen Knoten v aus G gibt, so dass v beim Durchlaufen von G mit Start in u nicht besucht wird, ist somit beschränkt durch (Summieren über alle Graphen aus $\mathcal{G}$ und alle möglichen Knotenpaare u, v):

$$\frac{n^2 \cdot |\mathcal{G}|}{(n^2 \cdot |\mathcal{G}|)^c} < 1.$$

Da die untersuchte Wahrscheinlichkeit der Wahrscheinlichkeit entspricht, dass die ausgewürfelte Folge keine universelle Durchlaufsequenz ist und diese Wahrscheinlichkeit kleiner als 1 ist, wissen wir, dass es eine universelle Durchlaufsequenz für $\mathcal{G}$ der behaupteten Länge gibt. □

Wir wollen nun die Schranke für d-reguläre Graphen konkretisieren: Für sie ist $|E| = dn/2$. Der Durchmesser eines Graphen ist definiert als

$$\max_{u \neq v} \text{dist}(u, v),$$

wobei $\text{dist}(u, v)$ die Länge eines kürzesten Weges von u nach v ist.
Man kann zeigen:
A) Die Anzahl gelabelter d-regulärer Graphen auf n Knoten ist beschränkt durch $(nd)^{O(nd)}$.
B) Der Durchmesser jedes zusammenhängenden d-regulären Graphen auf n Knoten ist $O(n/d)$.
Aus dem obigen Satz ergibt sich damit:
Für die Klasse der gelabelten zusammenhängenden d-regulären Graphen gibt es eine universelle Durchlaufsequenz der Länge $O(n^3 d \log n)$:
Der Durchmesser ist $O(n/d)$, daher ist $R(\mathcal{G}) = O(n/d)$, Einsetzen...
Es ist unbekannt, wie man so eine universelle Sequenz in Polynomialzeit oder noch besser in logarithmischem Platz berechnen kann.

4.8.2 Gerichtete Graphen

Das Problem STCON soll die Frage beantworten, ob es in einem gerichteten Graph einen gerichteten Weg von s nach t gibt.
Hier ist kein randomisierter Algorithmus bekannt, der ähnlich funktioniert wie bei den ungerichteten Graphen. Wir präsentieren einen randomisierten Algorithmus mit sehr schlechter Laufzeit, aber logarithmischem Platzbedarf für STCON.

Algorithmus:
Schritt 1: Starte in s und simuliere eine Irrfahrt der Länge $n-1$. (Anmerkung: Falls ein Knoten keine ausgehende Kante hat, verharre in diesem Knoten, ansonsten wähle einen der Nachbarn gemäß der Gleichverteilung.)
Falls der Knoten t erreicht wurde, gib JA aus. STOP.
Schritt 2: Wirf $\lceil n \log n \rceil$ Münzen. (Es kommt die 1 mit Wahrscheinlichkeit 1/2 und die 0 ebenso.) Falls alle Münzwürfe die 1 ergeben, STOP und NEIN ausgeben, ansonsten mache bei Schritt 1 weiter.

4.76 Satz. *Der gerade beschriebene Algorithmus hat logarithmischen Extraplatzbedarf und erwartete Rechenzeit $O(n^n)$. Außerdem gelten die folgenden Aussagen:*

a) Falls es keinen Weg von s nach t gibt, gibt der Algorithmus nie JA aus.

b) Falls es einen Weg von s nach t gibt, gibt der Algorithmus mit Wahrscheinlichkeit $p_{JA} \geq 1/2$ JA aus.

Beweis: Die Behauptungen über die Laufzeit und den benötigten Platz sind mehr oder weniger offensichtlich. (Das Zählen der Anzahl der gewürfelten Zufallsbits und das Speichern der aktuellen Position im Graphen geht auf logarithmischem Platz.)
Wir analysieren die Wahrscheinlichkeit p_{JA} und nehmen also an, dass es einen Weg von s nach t gibt.
Es gibt höchstens $n^{n-1} \leq n^n$ mögliche Irrfahrten der Länge $n-1$ von s aus. Mindestens eine davon findet den Knoten t. Damit ist die Wahrscheinlichkeit p_1, in Schritt 1 den Knoten t zu finden, mindestens $\frac{1}{n^n}$.
Also ist $p_{JA} \geq p_1 + (1-p_1) \cdot (1 - \frac{1}{n^n}) \cdot p_{JA}$.
Der erste Term steht dafür, dass man t im ersten Schritt findet, der zweite Term dafür, dass man t nicht im ersten Schritt findet und im zweiten Schritt entscheidet, dass man weitermacht und Knoten t später findet.
Die rechte Seite der Ungleichung ist eine Funktion, die monoton wachsend in p_1 ist. Wegen $p_1 \geq \frac{1}{n^n}$ folgt somit

$$p_{JA} \geq \frac{1}{n^n} + (1 - \frac{1}{n^n})^2 \cdot p_{JA}.$$

Da $(1-a)^2 \geq 1 - 2a$ ist, erhalten wir $p_{JA} \geq \frac{1}{n^n} + (1 - \frac{2}{n^n}) \cdot p_{JA}$, also $p_{JA} \cdot \frac{2}{n^n} \geq \frac{1}{n^n}$ und somit $p_{JA} \geq 1/2$. □

Übrigens könnte man auch auf die Idee kommen, folgenden Algorithmus zu entwerfen:
DO n^n mal: Schritt 1 des obigen Algorithmus.

Der Grund, warum man dies nicht machen kann, ist der, dass man für den Zähler, der bis n^n zählen muss, $\log n^n = n \log n$ viele Bits benötigt: Man käme also nicht mehr mit logarithmischem Platz aus.

4.9 Fingerprinting

Bezeichne $\mathbb{F}$ stets einen endlichen Körper.

In der Regel ist die Verifikation einer Lösung einfacher als ihre Berechnung. Beispielsweise ist das Lösen eines linearen Gleichungssystems schwieriger als das Testen mittels Einsetzen, ob es sich tatsächlich um eine gültige Lösung handelt.

Verifikation von Matrizenmultiplikationen
Jemand hat eine Software geschrieben, die das Produkt zweier $(n \times n)$-Matrizen $A \cdot B = C$ über dem Körper $\mathbb{F}$ berechnet. Kann man nun das Resultat schneller verifizieren als durch eine (unabhängige) zweite Multiplikation, von der wir wissen, dass sie das richtige Ergebnis liefert? Ja, dies funktioniert randomisiert.
Die Grundidee ist die folgende: Wenn $A \cdot B \neq C$, dann ist $(A \cdot B) \cdot x \neq C \cdot x$ für „viele“ x. Wenn hingegen $A \cdot B = C$, dann ist $(A \cdot B) \cdot x = C \cdot x$ für alle x.

Wir bemerken vorweg, dass ein Körper $\mathbb{F}$ immer die Elemente 0 und 1 enthält und dass somit ein Vektor $x \in \{0,1\}^n$ auch aus $\mathbb{F}^n$ ist. Zunächst eine technische Behauptung:

4.77 Behauptung. Sei $v \in \mathbb{F}^n$ ein Vektor, der nicht der Nullvektor ist und $z \in \mathbb{F}$. Wenn wir einen Spaltenvektor x gemäß der Gleichverteilung aus $\{0,1\}^n$ wählen, dann gilt:

$$\text{Prob}_x(v \cdot x = z) \leq 1/2.$$

Beweis: Da $v \neq 0$ ist, gibt es ein i mit $v_i \neq 0$. Sei o.B.d.A. $v_1 \neq 0$. Es sei also $v = (v_1, v')$ mit $v' = (v_2, \ldots, v_n)$ und $x = (x_1, x')$ mit $x' = (x_2, \ldots, x_n)$. Dann gilt $v \cdot x = v_1 x_1 + v' \cdot x'$ und somit

$$\begin{aligned} \text{Prob}_x(v \cdot x = z) &= \text{Prob}(x_1 = 0) \cdot \text{Prob}_{x'}(v' \cdot x' = z) + \text{Prob}(x_1 = 1) \cdot \text{Prob}_{x'}(v' \cdot x' = z - v_1) \\ &= \frac{1}{2} \cdot \big(\text{Prob}_{x'}(v' \cdot x' = z) + \text{Prob}_{x'}(v' \cdot x' = z - v_1)\big). \end{aligned}$$

Das $v' \cdot x' = z$ und $v' \cdot x' = z - v_1$ wegen $v_1 \neq 0$ zwei disjunkte Ereignisse sind, ist die Summe der beiden Wahrscheinlichkeiten in der Klammer kleiner gleich 1 und somit $\text{Prob}_x(v \cdot x = z) \leq 1/2$. □

4.78 Behauptung. Würfle $x \in \{0,1\}^n$ gemäß der Gleichverteilung aus. Wenn $A \cdot B \neq C$ ist, dann gilt $\text{Prob}_x((A \cdot B) \cdot x = C \cdot x) \leq \frac{1}{2}$.

Beweis: Für eine Matrix M bezeichne M_i die $i-$te Zeile in M. Wenn $M \cdot x$ der Nullvektor ist, dann ist $M_i \cdot x$ für alle i die Zahl Null. Somit gilt:

$$\text{Prob}_x((A \cdot B - C) \cdot x = 0) \leq \text{Prob}_x((A \cdot B - C)_i \cdot x = 0).$$

Wenn $A \cdot B \neq C$, dann gibt es eine Zeile i in $A \cdot B - C$, die nicht die Nullzeile ist. Für dieses i können wir $v := (A \cdot B - C)_i$ wählen und nun mit Hilfe der letzten Behauptung abschätzen, dass

$$\text{Prob}_x((A \cdot B - C)_i \cdot x = 0) \leq 1/2 \text{ ist.}$$

□

Eine Matrizenmultiplikation zweier $(n \times n)$-Matrizen benötigt nach der Schulmethode Laufzeit $O(n^3)$, nach Strassen $O(n^{2.7\cdots})$, bzw. neuere Ergebnisse erzielen eine Laufzeit von $O(n^{2.3\cdots})$. Mit der obigen Methode haben wir einen Algorithmus mit Fehlerwahrscheinlichkeit $\leq \frac{1}{2}$. Seine Laufzeit ist $O(n^2)$, weil man die Multiplikation $C \cdot x$ mit $O(n^2)$ Operationen berechnen kann und auch $(A \cdot B) \cdot x$ als $A \cdot (B \cdot x)$ berechnet.
Der Fehler lässt sich durch Iteration reduzieren.

Ein anderer Ansatz für die Reduktion der Fehlerwahrscheinlichkeit ist der, dass man die Einträge x nicht aus $\{0,1\}$, sondern aus $\mathbb{H} \subseteq \mathbb{F}$ wählt. Es lässt sich zeigen, dass dann die Fehlerwahrscheinlichkeit auf $\leq 1/|\mathbb{H}|$ sinkt.

Ein weiteres Beispiel: Die Multiplikation von zwei Polynomen $p_0x^0 + \cdots + p_nx^n$ und $q_0x^0 + \cdots + q_nx^n$. Die triviale Schulmethode benötigt hierfür Laufzeit $O(n^2)$, während die Fast Fourier Transformation (FFT) eine Multiplikation in Zeit $O(n \log n)$ erlaubt. Das Auswerten eines Polynoms auf Eingabe a erfordert mittels des Horner-Schemas Laufzeit $O(n)$.
Horner-Schema:

s`:=`p_n
`for` i`:=`$n-1$ `downto` 0 `do` s`=`$s \cdot a$`+`p_i

Die Grundidee ist die folgende: Wähle einen Körper $\mathbb{F}$ mit mindestens $2n+1$ Elementen und $\mathbb{H} \subseteq \mathbb{F}$, $|\mathbb{H}| \geq 2n+1$.
Zur Überprüfung, ob $p_1 \cdot p_2 = p_3$ ist, wähle gemäß der Gleichverteilung ein $r \in \mathbb{H}$ und gib „vermutlich gleich“ aus, falls $p_1(r) \cdot p_2(r) = p_3(r)$ und ansonsten „ungleich“.
Wenn $p_1 \cdot p_2 \neq p_3$ ist, dann ist das Polynom $q := p_1 \cdot p_2 - p_3$ nicht das Nullpolynom und es hat einen Grad, der durch $2n$ beschränkt ist. Wenn wir ein r auswürfeln, für das $p_1(r) \cdot p_2(r) = p_3(r)$ ist, dann ist dies eine Nullstelle von q, davon gibt es aber wegen der Gradbeschränkung höchstens $2n$ viele. Da $r \in \mathbb{H}$, mit $|\mathbb{H}| \geq 2n+1$, ist $\text{Prob}_r(p_1(r) \cdot p_2(r) = p_3(r)) \leq 2n/(2n+1) < 1$. Offensichtlich sollte man F und $\mathbb{H}$ möglichst groß wählen oder dieses Vorgehen mehrmals unabhängig iterieren, um die Wahrscheinlichkeit weiter zu reduzieren.
Beispiel: Es sei M eine $(n \times n)$-Matrix. Nach Definition ist die Determinante

$$\det(M) = \sum_{\pi \in S_n} \text{sgn}(\pi) M_{1\pi(1)} \cdots M_{n\pi(n)}.$$

Diese Summe besteht offenbar aus exponentiell vielen Termen. Determinanten können allerdings in $O(n^3)$ bzw. $O(n^{2.3\cdots})$ berechnet werden.

4.79 Definition (Vandermondesche Matrix)**.** Gegeben seien die Variablen $(x_1, \ldots, x_n)$, dann heißt $M = M(x_1, \ldots, x_n)$ mit

$$M = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix}$$

die Vandermondesche Matrix.

Wir bemerken am Rande, dass die Vandermondesche Matrix eine gewisse Rolle bei der Interpolation spielt. Sei zum Beispiel $f(x) = c_0 + c_1x + \cdots + c_{n-1}x_{n-1}$ ein Polynom, dann ist $M \cdot (c_0, \ldots, c_{n-1})^T$ der Vektor $(f(x_1), f(x_2), \ldots, f(x_{n-1}))^T$.
Die Determinante der Vandermondeschen Matrix ist ein multivariates Polynom. Für diese gilt:

4.80 Satz (Vandermonde)**.** *Für die Vandermondesche Matrix M gilt* $\det(M(x_1, \ldots, x_n)) = \prod_{j<i}(x_i - x_j)$.

Angenommen, wir hätten keinen Beweis dieser Aussage und wollten sie randomisiert überprüfen. Dann wollten wir $\det(M(x_1, \ldots, x_n)) - \prod_{j<i}(x_i - x_j)$ daraufhin überprüfen, ob es das Nullpolynom ist oder nicht.
Es gibt Aussagen über die Wahrscheinlichkeit, eine Nullstelle eines multivariaten Polynoms auszuwürfeln. Zunächst ein technisches Lemma, das wir später benutzen werden.

4.81 Lemma. *Seien E_1 und E_2 Ereignisse, dann gilt*

$$\textbf{Prob}\,(E_1) \leq \textbf{Prob}\left(E_1 \mid \overline{E_2}\right) + \textbf{Prob}\,(E_2)\,.$$

Beweis: Wenn $\mathbf{Prob}\left(\overline{E_2}\right) = 0$ ist, dann ist $\mathbf{Prob}\left(E_2\right) = 1$ und die Aussage folgt dann trivialerweise, egal, wie man das dann eigentlich undefinierte $\mathbf{Prob}\left(E_1 \mid \overline{E_2}\right)$ behelfsweise definiert. Sei also nun $\mathbf{Prob}\left(\overline{E_2}\right) > 0$.

$$\begin{aligned}\mathbf{Prob}\left(E_1\right) = \mathbf{Prob}\left(\left(E_1 \cap E_2\right) \cup \left(E_1 \cap \overline{E_2}\right)\right) &= \mathbf{Prob}\left(E_1 \cap E_2\right) + \mathbf{Prob}\left(E_1 \cap \overline{E_2}\right) \\ &\leq \mathbf{Prob}\left(E_2\right) + \mathbf{Prob}\left(E_1 \mid \overline{E_2}\right).\end{aligned}$$

Die erste Gleichung gilt, weil $E_1 \cap E_2$ und $E_1 \cap \overline{E_2}$ disjunkt sind und die Abschätzung für $\mathbf{Prob}\left(E_1 \cap \overline{E_2}\right)$ gilt, weil

$$\mathbf{Prob}\left(E_1 \mid \overline{E_2}\right) = \frac{\mathbf{Prob}\left(E_1 \cap \overline{E_2}\right)}{\mathbf{Prob}\left(\overline{E_2}\right)} \geq \mathbf{Prob}\left(E_1 \cap \overline{E_2}\right).$$

□

4.82 Satz (Schwartz-Zippel)**.** *Sei $Q(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ ein Polynom, das nicht das Nullpolynom ist. Sei d der Grad des Polynoms und $\mathbb{H} \subseteq \mathbb{F}$.*
Wenn wir einen Vektor $r = (r_1, \ldots, r_n) \in \mathbb{H}^n$ gemäß der Gleichverteilung auswählen, dann gilt:

$$\mathbf{Prob}\left(Q(r_1, \ldots, r_n) = 0\right) \leq \frac{d}{|\mathbb{H}|}.$$

Beweis: Durch Induktion über n. Für $n = 1$ liegt ein Polynom auf einer Variablen vor, welches maximal d Nullstellen hat, womit sofort $\mathbf{Prob}\left(Q(r_1) = 0\right) \leq \frac{d}{|\mathbb{H}|}$ folgt.
Induktionsschluss: Falls $Q(x_1, \ldots, x_n) = \text{const}$, so gilt die Aussage trivialerweise. Wenn Q nun kein konstantes Polynom ist, existiert eine Variable, von der das Polynom abhängt. Sei dies o.B.d.A. die Variable x_1. Also ist $Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i Q_i(x_2, \ldots, x_n)$, wobei $k \leq d$ und $k \geq 1$, da Q von x_1 abhängt. Die Zahl k sei so gewählt, dass Q_k nicht das Nullpolynom ist.
Die Wahl der Eingabe $r_1, \ldots, r_n$ können wir uns so vorstellen, dass zunächst $r_2, \ldots, r_n$ und dann r_1 gewählt werden.
Sei E_1 das Ereignis, dass $Q(r_1, \ldots, r_n) = 0$ ist und E_2 sei das Ereignis, dass (nach Fixierung von $r_2, \ldots, r_n$) der Wert $Q_k(r_2, \ldots, r_n) = 0$ ist.
Da Q_k nicht das Nullpolynom ist und Grad höchstens $d - k$ hat, folgt nach Induktionsvoraussetzung, dass $\mathbf{Prob}\left(E_2\right) \leq \frac{d-k}{|\mathbb{H}|}$ ist.
Wenn $Q_k(r_2, \ldots, r_n)$ nicht gleich Null ist, dann ist $q(x_1) := Q(x_1, r_2, \ldots, r_n)$ ein Polynom, das nicht das Nullpolynom ist und nur noch von einer Variablen abhängt. Wenn man den Induktionsanfang ($n = 1$) für q verwendet, dann erhält man daraus

$$\mathbf{Prob}\left(E_1 \mid \overline{E_2}\right) = \mathbf{Prob}\left(q(r_1) = 0\right) \leq \frac{k}{|\mathbb{H}|}.$$

Zusammen mit dem oben gezeigten Lemma erhalten wir

$$\begin{aligned}\mathbf{Prob}\left(Q(r_1, \ldots, r_n) = 0\right) &\leq \mathbf{Prob}\left(E_1 \mid \overline{E_2}\right) + \mathbf{Prob}\left(E_2\right) \\ &\leq \frac{k}{|\mathbb{H}|} + \frac{d-k}{|\mathbb{H}|} = \frac{d}{|\mathbb{H}|}.\end{aligned}$$

□

Hier eine kleine Randüberlegung zur Frage, ob man die obige Abschätzung verbessern könnte, wenn man vom Polynom Q und der Teilmenge $\mathbb{H}$ nur den Grad d und die Kardinalität von $\mathbb{H}$ kennt. Wir betrachten den Körper Z_p^* mit p prim und $p > d$. Das Polynom $(x_1 - 1) \cdot (x_1 - 2) \cdots (x_1 - d)$ hat den Grad d und d Nullstellen. Nun könnte es sein, dass $\mathbb{H} \supseteq \{1, \ldots, d\}$ ist, dann ist die Wahrscheinlichkeit, eine der Nullstellen auszuwürfeln, gerade $d/|\mathbb{H}|$, die Abschätzung kann also nicht verbessert werden.

4.9.1 Perfekte Matchings in Graphen

Ein Matching auf einem bipartiten Graphen $G = (V \cup U, E)$, $|U| = |V| = n$, ist eine Teilmenge E' der Kanten, die die Eigenschaft hat, dass keine zwei Kanten aus E' einen Knoten gemeinsam haben.
Das Matchingproblem besteht in der Aufgabe, ein möglichst großes Matching zu finden. Ein Matching im bipartiten Graphen heißt „perfektes Matching“, wenn es aus n Kanten besteht.
Das Matchingproblem auf bipartiten Graphen kann man mit Hilfe von Flussalgorithmen lösen. Die Laufzeit ist dann $O(m \cdot \sqrt{n}) = O(n^{2.5})$, wenn m die Kantenzahl des Graphen ist.

4.83 Satz (Edmonds Theorem)**.** *Sei A die $(n \times n)$-Matrix, die wie folgt definiert ist:*

$$A_{i,j} = \begin{cases} x_{i,j} & \textit{falls } (u_i, v_j) \in E \\ 0 & \textit{sonst.} \end{cases}$$

Diese Matrix wird auch Edmonds-Matrix genannt. Die Determinante $\det(A)$ *ist ein multivariates Polynom. G besitzt genau dann ein perfektes Matching, wenn* $\det(A)$ *nicht das Nullpolynom ist.*

Beweis: Wir bezeichnen zunächst für eine Permutation π mit $m(\pi)$ das Monom $x_{1\pi(1)} \cdots x_{n\pi(n)}$. Verschiedene Permutationen ergeben verschiedene Monome. Nach Definition ist

$$\det(A) = \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \cdot A_{1\pi(1)} \cdots A_{n\pi(n)}.$$

Für eine Permutation π ist der zugehörige Term $A_{1\pi(1)} \cdots A_{n\pi(n)} = m(\pi)$, wenn der Graph G alle Kanten $\{i, \pi(i)\}$ enthält. Ansonsten ist der Term Null.
Wenn $det(A)$ also nicht das Nullpolynom ist, dann gibt es eine Permutation π, so dass der Graph G alle Kanten $\{i, \pi(i)\}$ enthält. Diese Kanten bilden aber ein perfektes Matching.
Wenn ein perfektes Matching existiert, dann kann dieses als Permutation aufgefasst werden. Das entsprechende Monom $m(\pi)$ kommt in der Formel für $det(A)$ vor. Da für $\pi \neq \pi'$ die Monome $m(\pi) \neq m(\pi')$ sind, kann der Term nicht gegen ein anderes Monom gekürzt werden. Die Determinante ist also nicht das Nullpolynom. □

Ein randomisierter Algorithmus könnte folgendermaßen vorgehen: Wähle $\mathbb{F}$ und $\mathbb{H}$ groß genug und überprüfe für eine zufällig gewählte Belegung der Variablen in der A-Matrix mit Elementen aus $\mathbb{H}$, ob $\det(A(x)) = 0$ ist. Falls nein, gib „es existiert ein perfektes Matching“ aus, falls ja, gib „es existiert vermutlich kein perfektes Matching“ aus. Dieser Algorithmus kommt offensichtlich mit Laufzeit $O(n^{2.3\cdots})$ aus, da im Wesentlichen eine Determinante berechnet werden muss.

Gleichheitstest bei Strings

Alice hat Daten $a_0, \ldots, a_{n-1}$ mit $a_i \in \{0, 1\}$, Bob hat Daten $b_0, \ldots, b_{n-1}$ mit $b_i \in \{0, 1\}$.

Beide wollen feststellen, ob $a_i = b_i$ für alle $i \in \{0, \ldots, n-1\}$ ist, und zwar, indem Alice an Bob möglichst wenige Bits übermittelt. Man kann sich leicht überlegen, dass jeder deterministische Algorithmus im worst case mindestens n Bits übermitteln muss.
Wir betrachten hier einen randomisierten Algorithmus. Interpretiere die Daten als Zahlen:

$$A := \sum_{i=0}^{n-1} a_i \cdot 2^i, \quad B := \sum_{i=0}^{n-1} b_i \cdot 2^i.$$

Logischerweise sind die Daten genau dann identisch, wenn $A = B$ ist. Alice und Bob müssen also „nur noch“ $A = B$ überprüfen.

Algorithmus EQUALITYTEST
Sei $\tau := tn \cdot \ln(tn)$ für eine (geeignete) Konstante t. (Über t kann man die Fehlerwahrscheinlichkeit einstellen.)
Alice wählt gemäß der Gleichverteilung unter allen Primzahlen, die kleiner gleich τ sind, eine Primzahl p aus und sendet p sowie $A \bmod p$ an Bob. Bob überprüft, ob $A \equiv B \bmod p$.
Falls ja: „Daten sind (vermutlich) identisch".
Falls nein: „Daten verschieden".

Offensichtlich kann der Algorithmus Fehler begehen, aber nur dann, wenn er die Auskunft gibt „die Daten sind (vermutlich) identisch".

4.84 Satz. *EQUALITYTEST überträgt* $O(\log t + \log n)$ *Bits und irrt sich mit Wahrscheinlichkeit* $O(1/t)$.

Beweis: Da $p \leq \tau = tn \cdot \ln tn$ ist, kann man p und $A \bmod p$ mit $O(\log t + \log n)$ Bits darstellen. Wenn $\pi(x)$ die Anzahl der Primzahlen kleiner gleich x bezeichnet, dann ist die folgende Aussage aus der Zahlentheorie bekannt:

$$\text{Für alle } x \geq 17 \text{ gilt: } \frac{x}{\ln x} \leq \pi(x) \leq 1.26 \cdot \frac{x}{\ln x}.$$

Damit ist $\pi(tn \cdot \ln tn) = \Theta(tn)$.
Für wie viele Primzahlen p kann $A \equiv B \bmod p$ gelten, wenn $A \neq B$ ist?

$$A \equiv B \bmod p, \text{also } A - B \equiv 0 \bmod p, \text{also } |A - B| \equiv 0 \bmod p.$$

Da $A \neq B$, ist $|A-B| > 0$ und wegen $0 \leq A < 2^n$ und $0 \leq B < 2^n$ ist $|A-B| < 2^n$. Wie viele verschiedene Primzahlen kann eine natürliche Zahl $z < 2^n$ als Teiler haben? Natürlich höchstens n viele, denn das Produkt von n Primzahlen beträgt mindestens 2^n. Die Irrtumswahrscheinlichkeit des Algorithmus ist also beschränkt durch

$$\frac{n}{\pi(tn \cdot \ln tn)} = O(1/t).$$

□

Pattern Matching

Wir arbeiten in diesem Kapitel über dem Alphabet $\{0, 1\}$.
Gegeben ein String $X = (X_1, \ldots, X_n)$ und ein Muster $Y = (Y_1, \ldots, Y_m)$, dessen Vorkommen wir in X suchen.
Genauer: Wenn für $j \leq n - m + 1$ der String $X(j) = (X_j, \ldots, X_{m+j-1})$ den Teilstring von X der Länge m bezeichnet, der an Stelle j beginnt, dann suchen wir das kleinste j mit $X(j) = Y$ bzw. die Antwort, dass es ein solches j nicht gibt.
Offensichtlich kann ein trivialer Algorithmus in Laufzeit $O(n \cdot m)$ das Pattern Matching Problem lösen, aus der Vorlesung „Effiziente Algorithmen" ist aber auch ein Algorithmus (von Knuth, Morris und Pratt) bekannt, der das Problem in Laufzeit $O(n + m)$ löst.
Wir wollen skizzieren, wie man das Problem auch mit einem randomisierten Algorithmus angehen kann, der manchmal (in den Konstanten) effizienter ist bzw. mit weniger Speicherplatz auskommt.
Ein String $T = (T_1, \ldots, T_m)$ der Länge m wird als Zahl aufgefasst und bei gegebener Primzahl p können wir einen Fingerprint wie folgt definieren:

$$F(T) := \sum_{i=1}^{m} T_i \cdot 2^{m-i} \bmod p.$$

Wenn wir den Fingerprint des Teilstrings $X(j)$ kennen, dann müssen wir den Fingerprint des Teilstrings $X(j+1)$ nicht komplett neu berechnen. Der folgende Algorithmus nutzt das aus:

Algorithmus PatternMatch über dem Alphabet $\Sigma = \{0, 1\}$.
Berechne $F_Y := F(Y)$.
$j = 1$. Berechne $F_X := F(X(1))$. (*)
$wert := 2^{m-1} \bmod p$.
while $F_X \neq F_Y$ **and** $j \leq n - m$ **do**
begin
if $X_j = 1$ **then** $F_X := F_X - wert$.
$F_X := (2 \cdot F_X + X_{j+m}) \bmod p$. $j := j + 1$. (*)
end
if $F_X = F_Y$
then gib aus: „Y kommt (vermutlich) an Stelle j vor“. **STOP.**
else gib aus: „Y kommt in X nicht vor.“ **STOP.**

Der Algorithmus aktualisiert j und F_X so, dass an den Stellen (*) im Algorithmus gilt $F_X = F(X(j))$.
Der Algorithmus ist in dieser Form ein Monte-Carlo-Algorithmus.
Sei Q_j die 0-1-Zufallsvariable, die 1 ist, wenn $F(X(j)) = F(Y)$ ist, obwohl $X(j) \neq Y$ ist.
Wenn wir die Primzahl p gemäß der Gleichverteilung unter den Primzahlen wählen, die kleiner gleich $\tau = n^2 m \ln(n^2 m)$ sind, dann ist die Wahrscheinlichkeit, dass $Q_j = 1$ ist, durch $O(m/\pi(\tau))$ beschränkt. Eine Rechnung wie oben zeigt, dass also $\mathbf{Prob}\,(Q_j = 1) = O(1/n^2)$ ist.
Die Wahrscheinlichkeit, dass für irgendein j die Zufallsvariable $Q_j = 1$ ist, ist also durch $O(1/n)$ beschränkt.
Man kann den Algorithmus modifizieren, um einen Las-Vegas-Algorithmus zu bekommen.
Wenn nämlich die Meldung „...vermutlich...“ ausgegeben wird, dann kann man zunächst auf die triviale Art mit maximal m Vergleichen überprüfen, ob $Y = X(j)$ ist. Falls dies nicht der Fall ist, dann ist die Fehlersituation eingetreten und man muss die Schleife fortsetzen.
Wenn E die erwartete Anzahl an Stellen j mit $Q_j = 1$ bezeichnet, so errechnet sich die erwartete Laufzeit dieses Las-Vegas-Algorithmus als

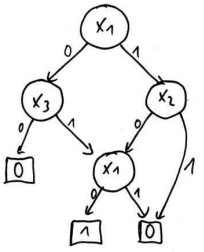
$$O(n + m) + E \cdot O(m) = O(n + m) + (1/n) \cdot O(m) = O(n + m).$$

Äquivalenztest bei FBDDs

FBDD steht abkürzend für „free binary decision diagram“. FBDDs sind spezielle BDDs.

4.85 Definition. Ein BDD ist ein gerichteter, azyklischer Graph. Es gibt zwei Arten von Knoten: Variablenknoten und Senken. Jeder Variablenknoten ist mit einer Variable aus $\{x_1, \ldots, x_n\}$ markiert, jede Senke mit einer Konstante aus $\{0, 1\}$. Variablenknoten haben einen 0-Nachfolger und einen 1-Nachfolger. Senken haben keine Nachfolger. Ein Knoten des BDDs wird als Quelle festgelegt. (In Zeichnungen typischerweise der Knoten mit Eingangsgrad 0.)

Das folgende Bild zeigt ein Beispiel für ein BDD.



Ein BDD legt zu jedem Input $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ auf kanonische Art und Weise einen Weg von der Quelle zu einer Senke fest. Wenn c die Markierung der erreichten Senke ist, so definiert man $f(a) := c$. Ein BDD legt also eindeutig eine boolesche Funktion fest.

4.86 Definition. Ein FBDD ist ein BDD, das zusätzlich folgende Eigenschaft hat: Auf jedem Weg von der Quelle zu einer Senke kommt jede Variable höchstens einmal vor.

Das Beispiel-BDD aus der Abbildung ist offensichtlich kein FBDD.

4.87 Definition. Das Äquivalenzproblem von FBDDs bezeichnet folgende Aufgabe:
Gegeben sind zwei FBDDs G_1 und G_2, die zwei boolesche Funktionen f_1 und f_2 darstellen. Ist $f_1 = f_2$, also $f_1(a) = f_2(a)$ für alle $a \in \{0,1\}^n$?

Es ist bislang kein deterministischer Polynomialzeitalgorithmus bekannt, der dieses Problem löst. Wir zeigen, dass es einen effizienten randomisierten Algorithmus für das Problem gibt. Zunächst halten wir fest, dass das Auswürfeln eines $a \in \{0,1\}^n$ und der Test, ob $f_1(a) = f_2(a)$ ist, nicht erfolgversprechend ist, weil sich f_1 und f_2 eventuell nur auf sehr wenigen Eingaben unterscheiden, so dass die Wahrscheinlichkeit, ein a mit $f_1(a) \neq f_2(a)$ auszuwürfeln, zu gering ist.
Wir wählen zunächst einen Körper $\mathbb{F}$, der mindestens $2n$ Elemente enthält. Zum Beispiel können wir Z_p^* für eine Primzahl $p \geq 2n$ wählen.
Die Art und Weise, wie ein FBDD G die boolesche Funktion darstellt, lässt sich auch anders beschreiben, und zwar bottom-up. Wir ordnen wie folgt jedem Knoten ein Polynom w zu:
$w_{\text{0-Senke}} = 0$, $w_{\text{1-Senke}} = 1$, und falls v ein Knoten mit 1-Nachfolger v_1, mit 0-Nachfolger v_0 und Variablenmarkierung x_i ist, so ist

$$w_v := x_i \cdot w_{v_1} + (1 - x_i) \cdot w_{v_0}.$$

Jeder Körper $\mathbb{F}$ enthält die Elemente $0, 1$ und -1, daher ist das Polynom als Polynom über dem Körper $\mathbb{F}$ wohldefiniert.
An der Quelle des FBDDs werde das Polynom w_G dargestellt.
Es ist klar, dass $w_G(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$ ist. Das Polynom w_G liefert also auf Eingaben aus $\{0,1\}^n$ genau die Funktionswerte der Funktion f. Wir erweitern die Funktion f zu einer Funktion

$$f^* : \mathbb{F}^n \to \mathbb{F} \text{ mit } f^*(r_1, \ldots, r_n) := w_G(r_1, \ldots, r_n).$$

Die beiden Funktionen f und f^* stimmen auf $\{0,1\}^n$ überein.
Der Definition von w_G sieht man direkt an, dass es sich um ein Polynom handelt, in dem jede Variable höchstens linear vorkommt. Dies liegt daran, dass auf jedem Weg jede Variable höchstens einmal vorkommt. Wir brauchen folgende Definition:

4.88 Definition. Sei $I \subseteq \{1, \ldots, n\}$ und $m_I := \prod_{i \in I} x_i$. Ein *multilineares Polynom* ist ein Polynom der Form

$$p(x_1, \ldots, x_n) = \sum_{I \subseteq \{1,\ldots,n\}} c_I \cdot m_I,$$

wobei die c_I Koeffizienten aus $\mathbb{F}$ sind.

Wir benötigen folgendes Lemma:

4.89 Lemma. *Sei p ein multilineares Polynom. Wenn es ein $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$ mit $p(r) \neq 0$ gibt, dann gibt es auch ein $a \in \{0,1\}^n$ mit $p(a) \neq 0$.*

Beweis: Sei $p(x_1, \ldots, x_n) = \sum_{I \subseteq \{1,\ldots,n\}} c_I \cdot m_I$. Es gibt offensichtlich mindestens ein I, so dass c_I von Null verschieden ist. Unter allen solchen I mit $c_I \neq 0$ wähle eines mit minimaler Kardinalität.
Wir beobachten nun, dass allgemein für einen Vektor $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ und eine Teilmenge $I' \subseteq \{1, \ldots, n\}$ der Wert $m_{I'}(a) = 1$ genau dann ist, wenn für alle $i \in I'$ der Wert $x_i = 1$ ist.

Wir wählen folgenden Input $a = (a_1, \ldots, a_n)$: $a_i := 1$, falls $i \in I$ und $a_i := 0$ sonst.

Damit ist $m_I(a) = 1$ und somit $c_I \cdot m_I(a) \neq 0$, aber, wie wir nun zeigen, $c_{I'} \cdot m_{I'}(a) = 0$ für alle $I' \neq I$. Der Grund: Für Teilmengen I' mit $|I'| \geq |I|$ ist offensichtlich $m_{I'}(a) = 0$ (es sind nicht genügend bzw. die falschen Positionen auf 1 gesetzt). Für Teilmengen I' mit $|I'| < |I|$ ist nach Wahl von I die Zahl $c_{I'} = 0$. Damit ist $p(a) = c_I \cdot m_I(a) \neq 0$. Der gewählte Vektor a hat also die gewünschte Eigenschaft. □

Im Beweis hat man also dafür gesorgt, dass das kürzeste aller Monome auf 1 gesetzt wird und alle anderen zu 0 auswerten.

4.90 Korollar. Sei p ein multilineares Polynom. Wenn $p(a) = 0$ für alle $a \in \{0,1\}^n$ ist, dann ist $p(r) = 0$ für alle $r \in \mathbb{F}^n$.

Wir beschreiben nun den Algorithmus für den Äquivalenztest und analysieren ihn dann.

Algorithmus FBDD-equivalence
Eingabe: Zwei FBDDs G_1 und G_2.
Berechne aus G_1 und G_2 die Polynome f_1^* und f_2^*.
Würfle $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$ gemäß der Gleichverteilung aus.
Berechne $f_1^*(r)$ und $f_2^*(r)$.
Ist $f_1^*(r) = f_2^*(r)$, so gib „vermutlich äquivalent“ aus.
Ist $f_1^*(r) \neq f_2^*(r)$, so gib „garantiert nicht äquivalent“ aus.

4.91 Satz. *Algorithmus FBDD-equivalence hat folgende Eigenschaften:*

a) Falls G_1 und G_2 äquivalent sind, dann gibt er „vermutlich äquivalent“ aus.

b) Falls G_1 und G_2 nicht äquivalent sind, dann gibt er mit Wahrscheinlichkeit mindestens $1/2$ „garantiert nicht äquivalent“ aus.

Beweis: a) Sei also $f_1 = f_2$, also $f_1(a) = f_2(a)$ für alle $a \in \{0,1\}^n$. Dann ist $f_1^*(a) = f_2^*(a)$ für alle $a \in \{0,1\}^n$.

Also ist $(f_1^* - f_2^*)(a) = 0$ für alle $a \in \{0,1\}^n$. Aus dem Korollar 4.90 folgt, dass auch $(f_1^* - f_2^*)(r) = 0$ für alle $r \in \mathbb{F}^n$ ist. (Die Differenz zweier multilinearer Polynome ist wieder ein multilineares Polynom). Der Algorithmus gibt also „vermutlich äquivalent“ aus.

b) Es gibt ein $a \in \{0,1\}^n$ mit $f_1(a) \neq f_2(a)$, also $f_1^*(a) \neq f_2^*(a)$.

Also gilt für das multilineare Polynom $d := f_1^* - f_2^*$, dass es nicht für alle $r \in \mathbb{F}^n$ den Wert $d(r) = 0$ liefert. Der Grad von d ist durch n beschränkt, da es nur die Variablen $x_1, \ldots, x_n$ gibt und es ein multilineares Polynom ist. Nach dem Satz von Schwartz-Zippel gilt für zufällig ausgewürfelte $r \in \mathbb{F}^n$:

$$\mathbf{Prob}\,(d(r) = 0) \quad \leq \quad \frac{\text{Grad von } d}{|\mathbb{F}|} \leq \frac{n}{2n} = 1/2.$$

Da der Algorithmus nur dann „vermutlich äquivalent“ ausgibt, wenn er ein r mit $d(r) = 0$ auswürfelt, folgt die Aussage aus b). □

4.10 Online-Algorithmen

Englisch	Deutsch
paging algorithm	Seitenwechselverfahren/-algorithmus
miss / fault	Fehler
hit	Treffer
oblivious adversary	nicht-adaptiver Widersacher/Gegner
evict a page	Seite auslagern

Bei den bisherigen Problemen war die Eingabe immer vollständig vorgegeben und der Algorithmus hat dazu eine entsprechende Ausgabe berechnet.
„Online“ bedeutet, dass die Eingabe erst „nach und nach“ enthüllt wird. Der Algorithmus muss jedoch sofort reagieren. Die Eingabe kann also als Folge von Anfragen gesehen werden.
Hier wollen wir so genannte Seitenwechselverfahren betrachten. Man kann sich vorstellen, dass eine 2-stufige Speicherhierarchie vorliegt.

- ein kleiner, schneller Speicher (cache)
- ein unendlich gedachter, langsamer Speicher (Festplatte)

Die vorliegende Datenmenge ist durch eine Menge von Seiten beschrieben. Wir nehmen an, dass der Cache k Seiten enthalten kann. Auf eine Anfragefolge $\rho_1, \rho_2, \ldots$, wobei für alle i die Zahl ρ_i eine Seite (bzw. Seitennummer) ist, muss der Seitenwechselalgorithmus wie folgt reagieren: Falls die angefragte Seite ρ_i im Cache ist, also ein „Treffer“ vorliegt, muss nichts getan werden.
Falls jedoch ρ_i nicht im Cache ist, ist dies ein „Fehler“ und die Seite muss vom langsamen Speicher eingelesen und in den Cache geholt („eingelagert“) werden. Es entstehen Zeitkosten, die wir als Kosten 1 veranschlagen.
Die Aufgabe des Seitenwechselalgorithmus ist es, die Anzahl der Fehler zu minimieren. Der Entscheidungsspielraum ist der, welche der Seiten aus dem Cache ausgelagert werden sollen.

Beispiele für (deterministische) Online-Seitenwechselalgorithmen sind:

- LRU (Least Recently Used). Lagere die Seite aus, nach der am längsten nicht mehr gefragt wurde. Wir nennen diese die „anfragenälteste Seite“.
- FIFO (First In First Out). Lagere die Seite aus, die am längsten im Cache ist.
- LFU (Least Frequently Used). Lagere die Seite aus, auf die am wenigsten zugegriffen wurde.
- LIFO (Last In First Out). Lagere die Seite aus, die als letzte eingelagert wurde.

Beispiel für Offline-Seitenwechselalgorithmus:
Ein Offline-Algorithmus kennt die gesamte Anfragefolge im Voraus.

Algorithmus MIN
Lagere die Seite aus, auf die am weitesten in der Zukunft wieder zugegriffen wird.
Genauer:
Zu Anfragefolge $\sigma_1, \ldots, \sigma_N$ und $t \in \{1, \ldots, N\}$ sowie eine Seite x definiere:

$$next_t(x) := \min_i \{\sigma_i = x \mid i \geq t+1\},$$

bzw. $next_t(x) := \infty$, falls die Menge, über die das Minimum berechnet wird, leer ist.

Sei X die Menge der Seiten, die sich im Cache befindet. Wenn auf der Anfrage σ_t ein Fehler vorliegt und es muss eine Auslagerung stattfinden, damit die angefragte Seite in den Cache passt, dann wähle die (bzw. eine) Seite $x \in X$, für die $next_t(x)$ maximal ist.

Bei Online-Algorithmen macht die klassische worst-case-Analyse keinen Sinn.

Beispiel: Zu jedem Online-Algorithmus kann man trivialerweise eine Anfragefolge konstruieren, so dass alle (bis auf k viele) Anfragen zu einem Fehler führen. Idee: Wähle die Anfragefolge $\sigma_i = i$ für $i = 1, \ldots, k$ und für $i \geq k{+}1$ wähle $\sigma_i := \{1, \ldots, k{+}1\} \setminus X_i$, wenn X_i die Menge der Seiten bezeichnet, die nach Anfrage i im Cache ist.
Um die Online-Algorithmen trotzdem bzgl. ihrer Güte einteilen zu können, braucht man eine *vergleichende* Analyse. Für den Vergleich zieht man den besten *Offline*-Algorithmus heran.
Ziel: Entwurf von Onlinealgorithmen, die nicht wesentlich schlechter sind als der optimale Offlinealgorithmus.
Gegeben sei eine Anfragefolge $\rho_1, \ldots, \rho_N$. Für einen deterministischen Online-Algorithmus A liegen seine Reaktionen auf die Folge fest. (Bei gegebenem Startzustand des Caches.) Daher definieren wir:

$$f_A(\rho_1, \ldots, \rho_N) := \text{die Kosten, die } A \text{ auf der Anfragefolge } \rho_1, \ldots, \rho_N \text{ entstehen.}$$

$$f_{opt}(\rho_1, \ldots, \rho_N) := \text{die Kosten, die dem besten Offline-Algorithmus auf der Anfragefolge } \rho_1, \ldots, \rho_n \text{ entstehen.}$$

4.92 Satz. *Der Offline-Seitenwechselalgorithmus* MIN *hat von allen Offline-Algorithmen die geringsten Kosten, ist also optimal.*

(Ohne Beweis.)

Ein Online-Algorithmus muss sich an diesem optimalen Algorithmus messen.

4.93 Definition. Ein Online-Seitenwechselalgorithmus A heißt *c-competitive*, wenn eine Konstante b existiert (die unabhängig von der Anfragelänge N ist, aber von k abhängen darf), so dass für jede Anfragefolge $\rho_1, \ldots, \rho_N$ gilt:

$$f_A(\rho_1, \ldots, \rho_N) \leq b + c \cdot f_{opt}(\rho_1, \ldots, \rho_N).$$

Als Gütekoeffizient c_A von A definiert man das kleinste solche c, d.h.

$$c_A := \inf\{c \mid A \text{ ist } c\text{-competitive}\}.$$

4.94 Satz. *LRU und FIFO sind k-competitive. LIFO und auch LFU sind für keine Konstante c-competitive.*

Beweis: Für die Aussage zu LIFO wähle die Anfragefolge $1, \ldots, k, k+1, k, k+1, k, \ldots$
LIFO produziert ab der $k+1$-ten Anfrage jedesmal einen Fehler, damit führt eine Anfragefolge der Länge N zu Kosten $N - k$, der optimale Algorithmus würde aber k und $k+1$ im Cache halten und spätestens ab Anfrage $k+1$ überhaupt keinen Fehler mehr produzieren, seine Kosten sind daher durch $k+1$ beschränkt und der Quotient der beiden Kosten ist durch keine Konstante nach oben beschränkt.
Die Aussage für FIFO beweisen wir nicht, aber die Aussage für LRU und LFU. Zunächst zur Behauptung zu LRU:
Gegeben sei eine Anfragefolge $\sigma_1, \ldots, \sigma_N$. Wir teilen diese in Phasen ein, und zwar betrachten wir die Anfragefolge von „rechts“, also von σ_N aus. Wir wählen das kleinste i, so dass die Anfragen von $\sigma_i, \ldots, \sigma_N$ höchstens k Fehler verursachen. Dann teilen wir die Anfragefolge $\sigma_1, \ldots, \sigma_{i-1}$ analog weiter in Phasen ein. Wir numerieren die Phasen von links nach rechts durch, von $P(0)$ bis $P(r)$.

Wir haben nun also die Eigenschaft, dass LRU in jeder Phase außer eventuell in der Phase $P(0)$ genau k Fehler hat. LRU macht also insgesamt höchstens $(r+1) \cdot k$ viele Fehler.

$P(0)$	$P(1)$	$\ldots$	$P(r)$	
x	$x \quad xx$	$x \;\; x$	$xx \;\; x$	LRU-Fehler

Wir zeigen, dass der optimale Algorithmus in jeder Runde $P(1)$ bis $P(r)$ mindestens einen Fehler macht. Da die Anzahl der LRU-Fehler höchstens $(r+1) \cdot k$ ist, während der optimale Algorithmus mindestens r Fehler macht, gilt

$$\#\text{ Fehler von LRU} \leq (k \cdot \#\text{ Fehler von OPT}) + k,$$

also ist LRU k-competitive.

Wir betrachten eine Runde $P(i)$ für $1 \leq i \leq r$ und nennen die Seite, die zuletzt vor Runde $P(i)$ angefragt wurde, p.
Behauptung: Runde $P(i)$ enthält Anfragen an mindestens k Seiten, die von p verschieden sind.

Aus der Behauptung folgt, dass der optimale Algorithmus in Runde $P(i)$ mindestens einen Fehler macht, denn: Zu Beginn von Runde $P(i)$ muss der optimale Algorithmus die Seite p im Cache haben, er enthält also höchstens $k-1$ Seiten, die von p verschieden sind. Während der Runde werden aber k Seiten angefragt, die von p verschieden sind, daher muss der optimale Algorithmus während der Runde mindestens eine Seite einlagern, also einen Fehler begehen.
Zum Beweis der Behauptung schauen wir uns die k Positionen an, an denen LRU einen Fehler macht und machen eine Fallunterscheidung:

A) An diesen Stellen werden k verschiedene Seiten nachgefragt, die von p verschieden sind. Dieser Fall ist trivial.

B) An einer dieser k Positionen wird nach p gefragt.

C) Weder Fall A), noch Fall B), also: es gibt eine Seite $q \neq p$, die mindestens zweimal angefragt wird.

Um die letzten beiden Fälle zu analysieren, zeigen wir zunächst folgende Aussage für LRU:

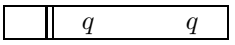
Sei q eine Seite und $t_1 < t_2, \quad \sigma_{t_1} = q, \sigma_{t_2} = q$ und auf σ_{t_2} hat LRU einen Fehler. Dann folgt daraus: die Folge $\sigma_{t_1}, \ldots, \sigma_{t_2}$ enthält mindestens $k+1$ Seiten.
Beweis: Zum Zeitpunkt t_1 wird auf q zugegriffen, d.h. q ist anschließend im Cache und damit die „anfragenjüngste“ Seite. Zum Zeitpunkt t_2 wird q nochmal in den Cache geladen, also muss q irgendwann dazwischen ausgelagert worden sein: $\exists t^*,\ t_1 < t^* < t_2$ mit: q wird zum Zeitpunkt t^* ausgelagert. Nach der LRU-Strategie muss q zum Zeitpunkt t^* die „anfragenälteste“ Seite gewesen sein. Damit das gilt, müssen bis zum Zeitpunkt t^* (inkl.) k Seiten ungleich q nachgefragt worden sein. Damit werden von t_1 bis t^* insgesamt $k+1$ Seiten nachgefragt. □

Fall B

p	p

Nach der obigen Aussage werden zwischen den Zugriffen auf p mindestens $k+1$ verschiedene Seiten nachgefragt, also mindestens k Seiten $\neq p$.

Fall C



Nach der obigen Aussage werden zwischen den Zugriffen auf q mindestens $k+1$ verschiedene Seiten nachgefragt, also mindestens k Seiten $\neq p$.

Insgesamt folgt: LRU ist k-competitive. □

4.95 Satz. *LFU ist für kein c c-competitive.*

Beweis: Wir geben eine Anfragefolge an, bei der LFU beliebig viele Fehler macht. Anfangs stehen $p_1, \ldots, p_k$ im Cache. Sei l eine beliebige natürliche Zahl.
Wähle

$$\sigma = \underbrace{p_1, \ldots, p_1}_{l \text{ mal}}, \underbrace{p_2, \ldots, p_2}_{l \text{ mal}}, \ldots, \underbrace{p_{k-1}, \ldots, p_{k-1}}_{l \text{ mal}}, \underbrace{p_k, p_{k+1}, p_k, p_{k+1}, \ldots, p_k, p_{k+1}}_{l-1 \text{ Paare}}$$

Sei t der Zeitpunkt, an dem zum ersten Mal auf p_{k+1} zugegriffen wird. Ab diesem Zeitpunkt lagert LFU abwechselnd p_k und p_{k+1} aus, da auf sie am wenigsten zugegriffen wurde. Optimal wäre es aber, zum Zeitpunkt t die Seite p_1 auszulagern. Damit gilt:
LFU produziert $2(l-1)-1$ Fehler, OPT produziert 1 Fehler.
Da wir l beliebig wählen können, kann LFU im Vergleich zu OPT beliebig schlechte Resultate liefern und ist daher nicht c-competitive für alle Werte von c. □

Wir schauen uns nun einen alternativen Beweis an für die Tatsache, dass LRU k–competitive ist, weil hier eine Methode verwendet wird, die bei Online–Algorithmen häufig zum Einsatz kommt.

Analyse von LRU mit Hilfe einer Potenzialfunktion

Man überlegt sich leicht, dass Algorithmus LRU immer die zuletzt nachgefragten k Seiten enthält.
Es liegt nahe, ein Alter für Seiten zu definieren, hier aus guten Gründen „rückwärts“:
Für eine Seite p im LRU-Cache beschreibt $w^{(t)}(p)$ das „Alter“ bzw. „Gewicht“ der Seite nach der t-ten Anfrage:
$w^{(t)}(p) = 1 \Leftrightarrow p$ ist die anfragenälteste Seite
$w^{(t)}(p) = k \Leftrightarrow p$ ist die anfragenjüngste Seite
$w^{(t)}(p) = i \Leftrightarrow p$ ist die i-t älteste Seite

Außerdem wählen wir $w^{(t)}(p) = 0$, falls die Seite nicht im LRU-Cache ist.
Der Einfachheit halber gehen wir davon aus, dass zu Beginn k Seiten im LRU-Cache sind, die das Alter 1 bis k haben. (Egal, wie).
Wenn eine Anfrage nach einer Seite mit Alter t stattfindet, dann „springt das Alter der Seite“ auf k, das Alter von jüngeren Seiten wird jeweils um 1 kleiner.
Wir betrachten nun den Inhalt des OPT-Caches genauer, wobei OPT der optimale Seitenwechselalgorithmus ist.
Beispiel: Es ist $k = 7$ und im OPT-Cache sind Seiten mit Gewicht $1, 3, 5, 0, 0, 0, 0$. Wenn eine Anfrage nach der Seite mit Gewicht 3 stattfindet, dann ist das Gewicht der Seiten im OPT-Cache anschließend $1, 7, 4, 0, 0, 0, 0$.
Findet nun eine Anfrage statt nach der in der Liste letzten Seite mit Gewicht 0, so sieht es im OPT-Cache anschließend so aus: $0, 6, 3, 0, 0, 0, 7$.
$W_{OPT}^{(t)}$ bezeichne die Summe der Gewichte der Seiten im Cache des Seitenwechselalgorithmus OPT nach Abarbeitung der t-ten Anfrage. Wir definieren nun eine Potenzialfunktion $\Phi^{(t)}$:

$$\Phi^{(t)} := \underbrace{k \cdot \ \#\ \text{OPT-Fehler}}_{\text{Term 3}} - \underbrace{\#\ \text{LRU-Fehler}}_{\text{Term 2}} + \underbrace{W_{OPT}^{(t)}}_{\text{Term 1}}.$$

Wir zeigen im Anschluss, dass Φ im Laufe der Zeit nicht kleiner wird, dass also $\Phi^{(t+1)} \geq \Phi^{(t)}$ für alle $t \geq 0$ gilt. Damit ist dann auch $\Phi^{(t)} \geq \Phi^{(0)}$ für alle $t \geq 0$. Da $W_{OPT}^{(t)} = O(k^2)$ ist, ist Term 1 durch eine Konstante beschränkt. Also gilt zu jedem Zeitpunkt:

$$\#\text{ LRU-Fehler} \leq k \cdot \#\text{ OPT-Fehler} + O(1),$$

also ist LRU k-competitive.

Bleibt also die Aufgabe, zu zeigen, dass Φ im Laufe der Zeit nur wachsen kann.
Bei einer Anfrage nach einer Seite v können vier verschiedene Fälle eintreten, die wir getrennt betrachten. Vorweg heben wir hervor, dass ein LRU-Fehler genau dann auftritt, wenn auf eine Seite mit Alter 0 zugegriffen wird.
Fall 1: Die Anfrage verursacht weder bei OPT noch bei LRU einen Fehler. Da somit das Alter einer Seite im OPT-Cache von t auf k springt und das Alter von maximal $k-t$ Seiten um eins kleiner wird, so wird W_{OPT} auf keinen Fall kleiner.
Fall 2: Die Anfrage nach v verursacht bei LRU einen Fehler, aber bei OPT keinen. Damit bleibt Term 3 unverändert, aber Term 2 wird um 1 größer. Wir zeigen nun, dass dies durch eine Veränderung in Term 1 ausgeglichen wird: Die Seite v trägt vor der Anfrage Gewicht 0 zu W_{OPT} bei, nach der Anfrage Gewicht k. Das Gewicht der anderen $(k-1)$ Seiten hat sich höchstens um 1 verkleinert. Damit ist W_{OPT} mindestens um 1 größer geworden.
Fall 3: Die Anfrage nach v verursacht bei OPT einen Fehler, aber bei LRU nicht. Dann wird Term 3 um k größer, Term 1 kann immer nur um k kleiner werden (da maximal von k Seiten das Alter um 1 kleiner wird.)
Fall 4: Die Anfrage nach v verursacht bei OPT einen Fehler und bei LRU auch. Dann wird Term 3 um k größer. Term 1 kann nur um maximal $k-1$ kleiner werden: Denn da LRU einen Fehler hat, wird auf eine Seite mit Alter 0 zugegriffen. Somit kann höchstens von $k-1$ Seiten das Alter um 1 kleiner werden. Schließlich wird Term 2 um 1 größer. Insgesamt erhalten wir, dass Φ nur wächst.

4.10.1 Eine untere Schranke für deterministische Online-Seitenwechselalgorithmen

Wir zeigen jetzt: jeder deterministische Online-Algorithmus, der c-competitive ist, hat $c \geq k$. Dies bedeutet im Prinzip, dass FIFO und LRU optimal sind.

Gegeben ein deterministischer Algorithmus A. Wir konstruieren eine Anfragefolge über $\{1, \ldots, k+1\}$. Seien zu Beginn $\{1, \ldots, k\}$ im Cache. Wähle $\sigma_1 = k+1$. Dann lagert Algorithmus A eine Seite p_1 aus. Wir wählen $\sigma_2 := p_1$, woraufhin Algorithmus A eine Seite p_2 auslagern muss. Wir wählen dann $\sigma_3 := p_2$, etc. Damit gilt, dass der deterministische Online-Algorithmus auf $\sigma_1, \ldots, \sigma_N$ in jedem Schritt einen Fehler macht, insgesamt also N Fehler.

Der optimale (Offline-)Algorithmus macht weniger Fehler, wie wir jetzt zeigen.
Wir benötigen folgende Definition: Für eine Anfragefolge $\sigma_1, \ldots, \sigma_N$ sei $T[i,j]$ die Anzahl verschiedener Seiten unter $\sigma_i, \ldots, \sigma_j$, also $T[i,j] := |\{\sigma_i, \ldots, \sigma_j\}|$.
Wir teilen die Anfragefolge in Runden ein. Dieses Vorgehen wollen wir als **Standardrundeneinteilung** bezeichnen:
Runde 1 beginnt mit Anfrage 1 und endet mit dem größten j, so dass $T[1,j] \leq k$ ist. Runde i beginnt zum Zeitpunkt $j^* :=$ „Ende der $(i-1)$-ten Runde“$+1$ und endet mit dem größten j, so dass $T[j^*, j] \leq k$ ist.

Behauptung OPT macht pro Runde höchstens einen Fehler.

Nach Konstruktion enthält die Anfragefolge nur Anfragen nach den $k+1$ Seiten $1, \ldots, k+1$. OPT lagert zu Beginn einer Runde i die Seite, die zu Beginn der Runde $i+1$ nachgefragt wird, aus

und hat dann die k Seiten, die in Runde i nachgefragt werden, im Cache. OPT macht daher nur Fehler auf der ersten Anfrage einer Runde.
Da jede Runde nach Konstruktion mindestens k Anfragen hat, macht der Online-Algorithmus mindestens k mal so viele Fehler.

Der Offline-Algorithmus kann als Widersacher aufgefasst werden, der dem Online-Algorithmus schwierige Aufgaben vorlegt.

4.10.2 Randomisierte Online-Algorithmen

Wir betrachten nun randomisierte Online-Algorithmen. Für diese machen drei verschiedene Widersachermodelle Sinn:

- der nicht-adaptive Widersacher
- der adaptive Online-Widersacher
- der adaptive Offline-Widersacher

Der nicht-adaptive Widersacher kennt unseren Online-Algorithmus und muss sich für eine Anfragefolge $\sigma_1, \ldots, \sigma_N$ entscheiden. Auf dieser Folge kann nun unser randomisierter Algorithmus starten. Der Widersacher kennt dabei unsere Zufallsentscheidung nicht. Die Fehler unseres Algorithmus werden gemessen gegen das Verhalten von $f_{OPT}(\sigma_1, \ldots, \sigma_N)$.

Der adaptive Widersacher allgemein kennt unseren Algorithmus und sieht auch unsere Zufallsentscheidungen. Er kann die Anfragen σ_i abhängig von der Reaktion unseres Algorithmus wählen.

Man unterscheidet den adaptiven Widersacher danach, ob er selbst sofort reagieren muss, also entscheiden muss, welche Seite möglicherweise ausgelagert wird, oder ob er selbst bis zum Ende der Anfragefolge warten darf:

Der adaptive Online-Widersacher muss nach Vorlage von σ_i selbst entscheiden, welche Seite eventuell ausgelagert wird.

Der adaptive Offline-Widersacher wartet bis zum Ende der (von ihm selbst) generierten Anfragenfolge $(\sigma_1, \ldots, \sigma_N)$ und startet dann darauf den optimalen Algorithmus.

Abhängig vom gewählten Widersachermodell gibt es unterschiedliche Definitionen des Begriffs c-competitive:

Für den nicht-adaptiven Widersacher muss es ein b geben, so dass für alle $\sigma_1, \ldots, \sigma_N$ gilt:

$$E(f_A(\sigma_1, \ldots, \sigma_N)) \leq b + c \cdot f_{OPT}(\sigma_1, \ldots, \sigma_N).$$

Beim adaptiven Online-Widersacher ist $\sigma_1, \ldots, \sigma_N$ jetzt eine zufällige Folge, die nach einer Wahrscheinlichkeitsverteilung P gewählt wird. Damit lautet die Definition: es gibt ein b mit:

$$E_{\sigma \in P}(f_A(\sigma_1, \ldots, \sigma_N)) \leq b + c \cdot E_{\sigma \in P}(f_w(\sigma_1, \ldots, \sigma_N))$$

Für den adaptiven Offline-Widersacher W gilt die Definition: es gibt ein b mit:

$$E_{\sigma \in P}(f_A(\sigma_1, \ldots, \sigma_N)) \leq b + c \cdot E_{\sigma}(f_{OPT}(\sigma_1, \ldots, \sigma_N))$$

Die Widersachermodelle wurden nach aufsteigender Mächtigkeit behandelt.

Es gilt:

- Algorithmus A ist c-competitive gegen den adaptiven Offline-Widersacher
  $\Rightarrow$ A ist c-competitive gegen den adaptiven Online-Widersacher
- Algorithmus A ist c-competitive gegen den adaptiven Online-Widersacher
  $\Rightarrow$ A ist c-competitive gegen den nicht-adaptiven Widersacher

4.10.3 Randomisierte Seitenwechselgorithmen gegen nicht-adaptive Widersacher

Wir zeigen: wenn ein randomisierter Seitenwechselalgorithmus A gegen nicht-adaptive Widersacher c-competitive ist, so gilt

$$c \geq H_k \approx \ln k,$$

wobei H_k die k-te harmonische Zahl ist.
Außerdem beschreiben wir einen randomisierten Seitenwechselalgorithmus MARKER, der $2H_k$-competitive ist gegen nicht-adaptive Widersacher. (Am Rande sei bemerkt, dass sogar randomisierte Algorithmen bekannt sind, die H_k-competitive sind.)

4.96 Satz. *Wenn ein randomisierter Seitenwechselalgorithmus A gegen nicht-adaptive Widersacher c-competitive ist, so gilt*

$$c \geq H_k \approx \ln k,$$

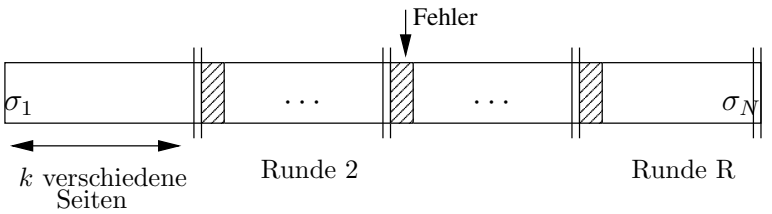
wobei H_k die k-te harmonische Zahl ist.

Beweis: Wir wenden Yaos Minimaxprinzip an. Es gibt nur endlich viele Seitenwechselalgorithmen. Dies kann man sich klar machen, indem man Seitenwechselalgorithmen formal darstellt als Funktion

$$f: \text{ Cacheinhalt } \times S \times \text{ Seite } \mapsto \text{ Cacheinhalt } \times S$$

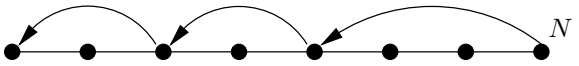
mit S, einer endlichen Zustandsmenge. Der Cacheinhalt und die Zahl der Seiten sind endlich, damit gibt es nur endlich viele Funktionen und damit Algorithmen.

Yaos Minimaxprinzip besagt nun:
Gegeben eine Wahrscheinlichkeitsverteilung P auf den Inputs (Anfragefolgen). Dann ist die beste erwartete Laufzeit, die man mit einem deterministischen Algorithmus unter P erzielen kann, eine untere Schranke für die beste erwartete worst-case-Laufzeit, die man mit einem randomisierten Algorithmus erzielen kann.
Wir wählen eine Wahrscheinlichkeitsverteilung auf den Inputs:
$\sigma_1 \stackrel{\wedge}{=}$ Seite aus $\{1, \ldots, k+1\}$ gemäß der Gleichverteilung.
$\sigma_{i+1} :=$ Seite aus $\{1, \ldots, k+1\} \setminus \{\sigma_i\}$ gemäß Gleichverteilung.
Es ist also verboten, die gleiche Seite zweimal hintereinander zu wählen.
Jeder randomisierte Online-Algorithmus macht pro Anfrage offensichtlich mit Wahrscheinlichkeit $1/k$ einen Fehler, hat also erwartete Fehleranzahl von N/k.
Nun schauen wir uns die erwartete Anzahl an Fehlern an, die der optimale (Offline-)Algorithmus macht. Zu diesem Zweck teilen wir die Anfragefolgen wieder gemäß der „Standardrundeneinteilung“ ein, dabei bezeichne R die Anzahl der so erhaltenen Runden. Wie oben gilt dann, dass der optimale Algorithmus höchstens R Fehler auf der vorliegenden Anfragefolge macht. Uns interessiert nun also der Erwartungswert von R.



Um diesen Erwartungswert zu ermitteln kann man sich auch fragen, wie groß die erwartete Länge einer Runde ist:
Die Auswahl der Folge $\sigma_1, \sigma_2, \ldots, \sigma_N$ entspricht einem random walk auf K_{k+1}, dem vollständigen Graphen auf $k+1$ Knoten. Da jede Runde mit dem Besuch der $k+1$-ten Seite beginnt, entspricht die Länge einer Runde der Anzahl Schritte, die man macht, bis man alle Knoten von K_{k+1} gesehen hat, minus 1.

Wir wissen, dass $\mathcal{C}(K_{k+1}) = k \cdot H_k$ ist (Behauptung 4.59).
Die erwartete Länge einer Runde ist also $k \cdot H_k - 1$.
Bei Anfragelänge N erwarten wir damit intuitiv, dass die erwartete Rundenzahl $\frac{N}{kH_k - 1}$ ist. Aber ist das so trivial zu zeigen? Es gilt ja zum Beispiel nicht immer für alle Zufallsvariablen X, dass $\mathbf{E}[1/X] = 1/\mathbf{E}[X]$ ist.
Eine probate Methode, um eine Aussage über die erwartete Rundenzahl zu bekommen ist die, auf die probabilistische Rekurrenz zurückzugreifen (siehe Abschnitt 3.1). Wir interpretieren die Position des Partikels als die Länge der Anfragefolge und wenn wir eine Runde mit h Seitenzugriffen haben, dann ist die Länge der Anfragefolge um h kleiner geworden, das Partikel befindet sich dann an Position $N - h$.



X_i ist die Zufallsvariable, die misst, wie viele Schritte man von Position i aus nach links springt. Wir haben oben $\mathbf{E}[X_i] = k \cdot H_k - 1$ gezeigt. Mit der Wahl von $g(i) := k \cdot H_k - 1$ ist g eine monotone Funktion, und wir können das Resultat für die probabilistische Rekurrenz verwenden: Es sei T_N die Anzahl der Schritte, bis man an Position 0 ist, wenn man von Position N aus startet. Dann ist

$$\mathbf{E}[T_N] \leq \sum_{i=1}^{N} \frac{1}{g(i)} = \frac{N}{kH_k - 1}.$$

Daraus folgt $\mathbf{E}[R] \leq \frac{N}{H_k - 1}$. Damit haben wir gezeigt, dass der optimale Algorithmus eine erwartete Fehlerzahl höchstens $\frac{N}{k \cdot H_k - 1}$ hat. Die erwartete Fehlerzahl jedes randomisierten Algorithmus ist $\frac{N}{k}$. Division ergibt, dass die Güte des randomisierten Algorithmus $\geq H_k$ ist. □

Algorithmus MARKER

MARKER benutzt pro Seite ein Markierungsbit $\in \{0, 1\}$. Zu Beginn sind alle Markierungsbits auf 0 gesetzt. (Invariante: Seiten außerhalb des Cache sind immer mit 0 markiert.) Das Bit einer angefragten Seite wird immer auf 1 gesetzt. Falls nun $k + 1$ Seiten mit 1 markiert sind, markiere die k Seiten im Cache mit 0. Falls eine Seite auszulagern ist, dann wähle unter allen Seiten im Cache nach der Gleichverteilung eine, die mit 0 markiert ist, und lagere diese aus.

Der Algorithmus teilt (implizit) die Anfragefolge in Runden ein. Zu Beginn einer Runde ist genau eine markierte Seite im Cache (im Schritt vorher waren alle Seiten im Cache markiert). Wir zeigen nun:

4.97 Satz. *Der MARKER-Algorithmus ist $2H_k$-competitive.*

Wir wollen nun die erwartete Fehlerzahl dieses Algorithmus analysieren. Wir schauen uns Seitenanfragen an, die Fehler verursachen. Sei σ_t eine Seite, die unmarkiert ist (auf die in dieser Runde noch nicht zugegriffen wurde) und die auch in der Vorgängerrunde nicht angefragt wurde. σ_t heißt dann „clean“. σ_t heißt „stale“ genau dann, wenn in der aktuellen Runde noch nicht auf σ_t zugegriffen wurde, aber in der Vorgängerrunde auf σ_t zugegriffen wurde. Wenn σ_t einen Fehler verursacht, dann ist σ_t clean oder stale.

Analyse Offline-Algorithmus
Wir analysieren nun den Offline-Algorithmus. Wir greifen uns eine Runde heraus: l sei die Anzahl der Anfragen, die clean in dieser Runde sind. Sei S_{off} die Seitenmenge, die der Offline-Widersacher im Cache hat und S_{marker} die Seitenmenge, die der Marker-Algorithmus im Cache hat. Weiterhin sei $d_{\text{Init}} := |S_{\text{off}} \backslash S_{\text{marker}}|$ zu Beginn der betrachteten Runde und $d_{\text{final}} := |S_{\text{off}} \backslash S_{\text{marker}}|$ am Ende der betrachteten Runde.

P.S.: $|S_{\text{off}} \backslash S_{\text{marker}}| = |S_{\text{marker}} \backslash S_{\text{off}}|$, weil beide Kardinalität k haben.
M_{off} sei die Anzahl der Fehler, die der Offline-Algorithmus in der Runde macht.

1.) $M_{\text{off}} \geq l - d_{\text{Init}}$.
Der Marker-Algorithmus macht in der betrachteten Runde mindestens l Fehler. Von diesen l Seiten kann der Offline-Widersacher höchstens d_{Init} viele Seiten im Cache haben.

2.) $M_{\text{off}} \geq d_{\text{final}}$.
Die d_{final} Seiten, die am Ende der Runde im Marker-Cache drin sind, müssen irgendwann während der Runde angefragt worden sein. Damit müssen sie irgendwann während der Runde im Offline-Cache gewesen sein, am Schluss aber sind sie dort nicht mehr. Also hat es mindestens d_{final} Auslagerungen, also Fehler, gegeben.

Damit folgt

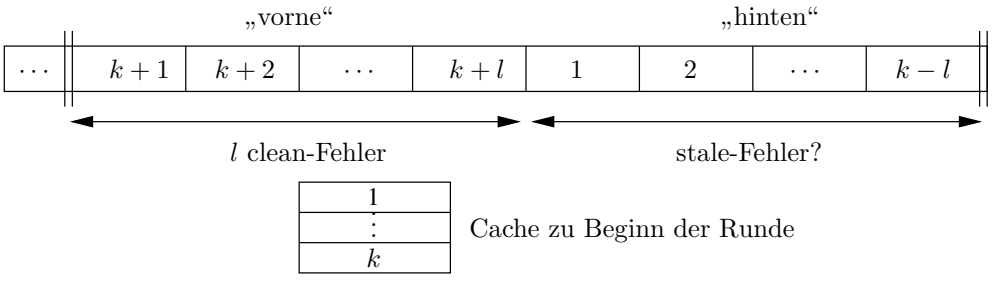
$$\begin{aligned} M_{\text{off}} &\geq \max\{d_{\text{final}}, l - d_{\text{Init}}\} \\ &\geq \frac{d_{\text{final}} + l - d_{\text{Init}}}{2} \qquad \text{(Maximum ist} \geq \text{Durchschnitt)} \end{aligned}$$

Sei l_j das l für die j-te Runde. Dann folgt für die Fehlerzahl

$$\begin{aligned} \text{Fehlerzahl} &\geq \sum_{j=1}^{\text{Rundenzahl}} \frac{l_j + d^j_{\text{final}} - d^j_{\text{Init}}}{2} \\ &\geq \frac{1}{2} \sum_{j=1}^{\text{Rundenzahl}} l_j + (d^{\text{Rundenzahl}}_{\text{final}} - d^1_{\text{Init}}) \\ &\geq \frac{1}{2} \sum_{j=1}^{\text{Rundenzahl}} l_j \end{aligned}$$

Wir haben damit gezeigt, dass die erwartete Fehlerzahl des Offline-Algorithmus $\geq \frac{1}{2} \sum_{j=1}^{\text{Rundenzahl}} l_j$ ist.

Analyse Marker-Algorithmus
Wir analysieren nun die Fehler des Marker-Algorithmus. Dafür greifen wir uns wieder eine Runde heraus. Der Algorithmus macht l Fehler auf den clean-Zugriffen. Es gibt aber auch Fehler bei Zugriffen auf stale-Seiten! Die Wahrscheinlichkeit, einen stale-Fehler zu haben, wird maximiert, wenn die clean-Zugriffe alle zuerst stattfinden. Dieser Fall ist der worst-case.



Beispielhaft schauen wir uns für die Seite 2 an: Mit welcher Wahrscheinlichkeit passiert ein Fehler beim Zugriff auf die 2?
Wir unterscheiden drei Fälle:

A) 1 und 2 wurden vorne rausgekegelt (ausgelagert).

B) 2 wurde vorne rausgekegelt, 1 nicht.

C) 2 wurde vorne nicht rausgekegelt, aber die 1 und dann wurde die 2 rausgekegelt beim Einlagern der 1.

Die drei Fälle haben die folgenden Wahrscheinlichkeiten:

$$\text{A) } \frac{\binom{k-2}{l-2}}{\binom{k}{l}} \qquad \text{B) } \frac{\binom{k-2}{l-1}}{\binom{k}{l}} \qquad \text{C) } \frac{\binom{k-2}{l-1}}{\binom{k}{l}} \cdot \frac{1}{k-l}$$

$$\textbf{Prob}\,(A \cup B \cup C) = \frac{\binom{k-2}{l-2}}{\binom{k}{l}} + \frac{\binom{k-2}{l-1}}{\binom{k}{l}} + \frac{\binom{k-2}{l-1}}{\binom{k}{l}} \cdot \frac{1}{k-l} = \cdots = \frac{l}{k-1}.$$

Für Element i statt 2 ergibt sich (mit einer längeren Rechnung)

$$\frac{l}{k-i+1}.$$

Die erwartete Anzahl an Fehlern in einer Runde ist $\leq l + \frac{l}{k} + \frac{l}{k-1} + \frac{l}{k-2} + \cdots + \frac{l}{l+1} \leq l \cdot H_k$. Wir summieren über alle Runden. Die erwartete Fehleranzahl ist nun $\leq H_k \sum_{j=1}^{\text{Rundenzahl}} l_j$.

Der optimale Algorithmus hatte mindestens $(1/2) \cdot \sum_{j=1}^{\text{Rundenzahl}} l_j$ viele Fehler, der Quotient aus beiden Fehlerzahlen ist $\leq 2H_k$. □

Anhang A

Ergänzungen

Eine einseitige Variante von Tschebyscheff

Die Abschätzung von Tschebyscheff gibt es auch in einer einseitigen Variante, die auch manchmal als Ungleichung von Tschebyscheff-Cantelli bezeichnet wird. Sie besagt folgendes:

A.1 Satz. *Sei X eine reellwertige Zufallsvariable und $\mathbf{V}[X]$ die Varianz von X. Wenn $\mathbf{V}[X] > 0$ ist, dann gilt für alle $t \geq 0$:*

$$\mathbf{Prob}\left(X - \mathbf{E}[X] \geq t \cdot \sqrt{\mathbf{V}[X]}\right) \leq \frac{1}{1+t^2}.$$

Beweis: Für $t = 0$ gilt die Aussage trivialerweise, sei also $t > 0$ im folgenden Wir definieren die Zufallsvariable $Y := X - \mathbf{E}[X]$, die als Erwartungswert $\mathbf{E}[Y] = 0$ und als Varianz $\mathbf{V}[Y] = \mathbf{V}[X]$ hat. Außerdem ist $\mathbf{V}[Y] = \mathbf{E}\left[Y^2\right]$.
Wir wollen für ein gegebenes $T > 0$ die Wahrscheinlichkeit analysieren, dass $Y \geq T$ ist. Dazu definieren wir zunächst eine Indikatorvariable $I_{Y \geq T}$, die den Wert 1 annimmt, wenn $Y \geq T$ ist und sonst gleich 0 ist. Damit ist $\mathbf{Prob}(Y \geq T) = \mathbf{E}[I_{Y \geq T}]$.
Für alle $c > 0$ ist $\frac{(Y+c)^2}{(T+c)^2} \geq I_{Y \geq T}$, denn $\frac{(Y+c)^2}{(T+c)^2}$ ist immer größer oder gleich 0 und wenn $I_{Y \geq T}$ den Wert 1 annimmt, dann hat der Quotient $\frac{(Y+c)^2}{(T+c)^2}$ den Wert mindestens 1. Also ist

$$\begin{aligned}
\mathbf{Prob}(Y \geq T) = \mathbf{E}[I_{Y \geq T}] &\leq \mathbf{E}\left[\frac{(Y+c)^2}{(T+c)^2}\right] \\
&= \frac{\mathbf{E}\left[(Y+c)^2\right]}{(T+c)^2} \\
&= \frac{\mathbf{E}\left[Y^2 + 2cY + c^2\right]}{(T+c)^2} \\
&= \frac{\mathbf{V}[Y] + c^2}{(T+c)^2}.
\end{aligned}$$

Wir wählen nun $c := \mathbf{V}[Y]/T > 0$ und setzen ein:

$$\begin{aligned}
\mathbf{Prob}(Y \geq T) &\leq \frac{\mathbf{V}[Y] + (\frac{\mathbf{V}[Y]}{T})^2}{(T + \frac{\mathbf{V}[Y]}{T})^2}. \\
&= \frac{\mathbf{V}[Y]}{T} \cdot \frac{T + \frac{\mathbf{V}[Y]}{T}}{(T + \frac{\mathbf{V}[Y]}{T})^2} \\
&= \frac{\mathbf{V}[Y]}{T^2 + \mathbf{V}[Y]}.
\end{aligned}$$

Setzen wir nun $T := t \cdot \sqrt{\mathbf{V}[Y]} > 0$ ein, so erhalten wir die Aussage. □

Übrigens würde der Satz falsch, wenn wir $\mathbf{V}[X] = 0$ zuließen, denn dann hätten wir eine Zufallsvariable, die mit Wahrscheinlichkeit 1 den Wert $X = \mathbf{E}[X]$ annimmt und es wäre $\mathbf{Prob}\left(X - \mathbf{E}[X] \geq t \cdot \sqrt{\mathbf{V}[X]}\right) = \mathbf{Prob}(0 \geq 0) = 1$ und die Aussage gälte nur für $t = 0$.

Bälle und Boxen

A.2 Satz. *Gegeben sind n Boxen, man macht m Ballwürfe und $E_{\geq k}$ sei das Ereignis, dass Box 1 mindestens k Bälle abbekommt. Dann gilt:*

$$\mathbf{Prob}(E_{\geq k}) \leq \binom{m}{k} \cdot \left(\frac{1}{n}\right)^k.$$

Beweis: Jede solche Folge von Ballwürfen kann man als Tupel $(w_1, \ldots, w_m)$ mit $w_i \in \{1, \ldots, n\}$ ansehen.
Für eine Teilmenge $A \subseteq \{1, \ldots, m\}$ sei E_A das Ereignis, dass $w_i = 1$ für alle $i \in A$ ist. Es ist klarerweise $\mathbf{Prob}(E_A) = (1/n)^{|A|}$. Nun ist:

$$\begin{aligned}
\mathbf{Prob}(E_{\geq k}) &= \mathbf{Prob}\left(\bigcup_{A,\ k\text{-elementig}} E_A\right) \\
&\leq \sum_{A,\ k\text{-elementig}} \mathbf{Prob}(E_A). \\
&= \binom{m}{k} \cdot \left(\frac{1}{n}\right)^k.
\end{aligned}$$

□

Einschub zu „zweistufigen Experimenten“

Dem einen oder anderen mag dieser Einschub sicherlich als Haarspalterei vorkommen, aber er ist ja auch nur als Warnung für alle kritischen Leser/Mitdenker gedacht.

Angenommen, man macht folgendes Experiment: Zunächst wirft man eine Münze $w \in \{0, 1\}$. Falls $w = 0$ ist, so wirft man einen zwölfseitigen Würfel W_{12}, falls $w = 1$ ist, so wirft man einen sechsseitigen Würfel W_6. Wenn W die Zahl ist, die wir zu sehen bekommen, was ist dann $E[W]$? Intuitiv erwarten wir als Erwartungswert $(1/2) \cdot 3.5 + (1/2) \cdot 5.5$. Der folgende Einschub zeigt uns, dass unsere Intuition korrekt ist, dass wir aber in bestimmten Situationen aufpassen müssen.
X sei eine Zufallsvariable mit Trägermenge $\{1, \ldots, n\}$. $T_1, \ldots, T_n$ seien Zufallsvariablen mit Trägermengen $M_1, \ldots, M_n$, wobei die M_i nur positive (reelle) Zahlen enthalten dürfen und abzählbar sein sollen. Wir können also $M_i = \{w_{i,1}, w_{i,2}, \ldots\}$ schreiben. Sei außerdem $\mathbf{E}[T_i] < \infty$. Die zweistufige Zufallsvariable T_X sei beschrieben durch

$$\mathbf{Prob}(T_X = w_{i,j}) := \mathbf{Prob}(X = i) \cdot \mathbf{Prob}(T_i = w_{i,j}).$$

A.3 Lemma. *Sei für alle i die Konstante $F_i := \mathbf{E}[T_i]$ definiert. Dann ist $\mathbf{E}[T_X] = \mathbf{E}[F_X]$.*

Beweis: (Skizze). Die Definition des Erwartungswerts verlangt, dass wir alle Elemente x aus der Trägermenge durchlaufen, x mit der Wahrscheinlichkeit des Auftretens von x multiplizieren und das Ganze aufsummieren. Wenn wir es mit einer unendlichen Trägermenge zu tun haben, dann haben wir es also mit einer unendlichen Reihe zu tun. Wir schreiben hin:

$$\begin{aligned}\mathbf{E}[T_X] &= w_{1,1}\cdot\mathbf{Prob}(X=1)\cdot\mathbf{Prob}(T_1=1)+\cdots+w_{n,1}\cdot\mathbf{Prob}(X=n)\cdot\mathbf{Prob}(T_n=1)+\\ &\quad +w_{1,2}\cdot\mathbf{Prob}(X=1)\cdot\mathbf{Prob}(T_1=2)+\cdots+w_{n,2}\cdot\mathbf{Prob}(X=n)\cdot\mathbf{Prob}(T_n=2)+\\ &\quad +\cdots\end{aligned}$$

Diese Reihenfolge der Summanden wählen wir, da klar ist, dass in dieser Aufzählung jedes $w_{i,j}$ irgendwann mal vorkommt.
Wir würden nun gerne ohne großes Nachdenken schreiben, dass dieses gleich den folgenden Formeln ist:

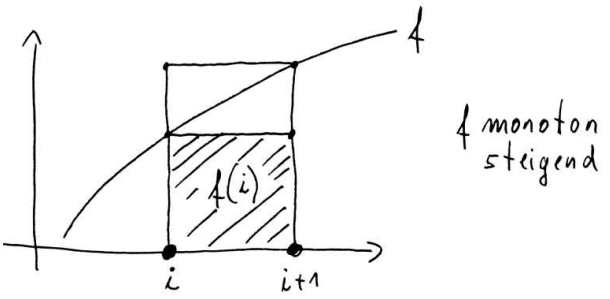
$$\begin{aligned}\mathbf{E}[T_X] &= (w_{1,1}\cdot\mathbf{Prob}(X=1)\cdot\mathbf{Prob}(T_1=1)+w_{1,2}\cdot\mathbf{Prob}(X=1)\cdot\mathbf{Prob}(T_1=2)+\cdots)\\ &\quad +(w_{2,1}\cdot\mathbf{Prob}(X=2)\cdot\mathbf{Prob}(T_2=1)+w_{2,2}\cdot\mathbf{Prob}(X=2)\cdot\mathbf{Prob}(T_2=2)+\cdots)\\ &\quad +\cdots\\ &= \mathbf{Prob}(X=1)\cdot\mathbf{E}[T_1]+\cdots+\mathbf{Prob}(X=n)\cdot\mathbf{E}[T_n]\\ &= \mathbf{Prob}(X=1)\cdot F_1+\cdots+\mathbf{Prob}(X=n)\cdot F_n\\ &= \mathbf{E}[F_X].\end{aligned}$$

Wenn wir das aber tun, dann haben wir die Summanden umgruppiert, und wie wir alle wissen (sollten), kann man das bei unendlichen Reihen nicht immer tun. Ich erinnere zum Beispiel daran, dass die Reihe $\sum_{i=0}^{\infty}(-1)^i$ nicht konvergiert, aber bei naivem Gruppieren könnte man ja auf die Idee kommen, diese Reihe als $(1-1)+(1-1)+(1-1)+\cdots=0$ zu schreiben.
Wann darf man umgruppieren? Wenn die Reihe absolut konvergent ist. Und damit haben wir in unserem konkreten Fall keine Probleme, da die Trägermengen unserer T_i nur positive Zahlen enthalten, die Wahrscheinlichkeiten alle nicht negativ sind und auch noch $\mathbf{E}[T_i]<\infty$ ist. Wer dies nun komplett und formal beweisen möchte, dem sei dies als Übungsaufgabe empfohlen. □

Summen durch Integrale abschätzen

Angenommen, wir haben eine monoton steigende Funktion f vorliegen, deren Integral durch die Stammfunktion F beschrieben ist (es gilt also $F'=f$).
Wir zeigen nun, wie man die diskrete Summe $\sum_{i=1}^{n} f(i)$ mit Hilfe der Stammfunktion F abschätzen kann.
Da f monoton steigend ist, kann man für alle $i=1,\ldots,n$ folgende Abschätzung einsehen (siehe auch Bild):

$$f(i)\le\int_i^{i+1} f(x)\,dx\le f(i+1).$$



Damit ist $f(1)+\cdots+f(n-1) \leq \int_1^n f(x)\ dx \leq f(2)+\cdots+f(n)$ und wir erhalten die Abschätzungen

$$f(1) + \int_1^n f(x)\ dx \leq f(1) + \cdots + f(n) \leq f(n) + \int_1^n f(x)\ dx.$$

Natürlich erhalten wir auf vollkommen analoge Art die folgende Abschätzung, wenn f eine monoton fallende Funktion ist:

$$f(n) + \int_1^n f(x)\ dx \leq f(1) + \cdots + f(n) \leq f(1) + \int_1^n f(x)\ dx.$$

Wir wollen diese Abschätzung an einigen Beispielen verwenden:

Die harmonischen Zahlen

A.4 Satz. *Für die n-te harmonische Zahl* $H(n) := (1/1) + (1/2) + \cdots + (1/n)$ *gilt:*

$$(1/n) + \ln n \leq H(n) \leq 1 + \ln n.$$

Beweis: Die Funktion $f(x) = 1/x$ ist monoton fallend und hat als Stammfunktion $F(x) = \ln x$. Da also $\int_1^n (1/x)\ dx = \ln n - \ln 1 = \ln n$ ist, erhalten wir wegen $f(1) = 1$ und $f(n) = (1/n)$ die Abschätzung aus dem Satz. □

Eine Abschätzung für $\log n!$

Wir analysieren zunächst $\ln(n!)$. Wegen $\log x = \ln x / \ln 2$ erhalten wir auch eine Abschätzung für $\log(n!)$. Es ist $\ln n! = \ln(n \cdot (n-1) \cdots 2 \cdot 1) = \ln n + \ln(n-1) + \cdots + \ln 1$. Wir wählen also $f(x) = \ln x$. Dies ist eine monoton steigende Funktion, die als Stammfunktion die Funktion $F(x) = x \cdot \ln x - x$ hat.
Somit ist $\int_{i=1}^n \ln x\ dx = F(n) - F(1) = n \cdot \ln n - n + 1$.
Wir erhalten als Abschätzung:

$$n \cdot \ln n - n + 1 \leq \ln n! \leq n \cdot \ln n - n + 1 + \ln n.$$

Und somit als Abschätzung für $\log n!$:

$$n \cdot \log n - (n/\ln 2) + (1/\ln 2) \leq \log n! \leq n \cdot \log n - (n/\ln 2) + (1/\ln 2) + \log n.$$

Wir haben somit gezeigt:

A.5 Satz. $\log n! = n \log n - (1/\ln 2) \cdot n + O(\log n)$.

Wir bemerken, dass $1.442 \leq 1/\ln 2 \leq 1.443$ ist.

Summen von Polynomen

Sei $f(x) = x^k$ für eine reelle Zahl $k \geq 0$. Dann ist f für $x \geq 0$ eine monoton steigende Funktion mit Stammfunktion $F(x) = \frac{1}{k+1} \cdot x^{k+1}$. Somit ist $\int_1^n x^k\ dx = \frac{1}{k+1} \cdot n^{k+1} - \frac{1}{k+1}$. Wir können abschätzen:

$$\frac{1}{k+1} \cdot n^{k+1} - \frac{1}{k+1} + 1 \leq \sum_{i=1}^n i^k \leq \frac{1}{k+1} \cdot n^{k+1} - \frac{1}{k+1} + n^k.$$

Damit sehen wir sofort ein, dass $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$ ist, also zum Beispiel

$$\begin{aligned}\sum_{i=1}^{n} i &= \Theta(n^2)\\ \sum_{i=1}^{n} i^{2.5} &= \Theta(n^{3.5})\\ \sum_{i=1}^{n} i^3 &= \Theta(n^4).\end{aligned}$$

Als Übungsaufgabe kann man es nun dem Leser oder der Leserin überlassen, die Summe $\sum_{i=1}^{n} i^k$ abzuschätzen, wenn k negativ und von -1 verschieden ist.

Eine Limesaussage

Bei der Analyse des Coupon-Collector-Theorems benötigen wir folgende Aussage, die wir hier ohne Beweis wiedergeben:
Sei n eine natürliche Zahl und c eine reelle Zahl. Dann gilt für $m = n \cdot \ln n + cn$ die folgende Aussage:

$$\lim_{n\to\infty} \binom{n}{k} \cdot (1 - \frac{k}{n})^m = \frac{e^{-ck}}{k!}.$$

Zur Chernoffschranke

A.6 Satz. *Sei*

$$f(x) := (1 + x) \cdot \ln(1 + x) - x - cx^2.$$

Dann gilt:

- *Wenn $c = (1/2)$ ist, dann gilt $f(x) \geq 0$ für alle $x \in (-1, 0]$.*
- *Wenn $c = 2\ln 2 - 1 \approx 0.386$ ist, dann gilt $f(x) \geq 0$ für alle $x \in [0, 1]$.*

Beweis: f hat als Ableitung $f'(x) = \ln(1 + x) - 2cx$. Da $\ln(1 + x) \leq x$ für alle $x > -1$ ist, ist $f'(x) \leq x(1 - 2c)$ für alle $x > -1$. Für $c = 1/2$ ist also $f'(x) \leq 0$. Damit ist f auf dem Intervall $(-1, 0]$ monoton fallend und hat sein Minimum auf dem Intervall $(-1, 0]$ an der Stelle $x = 0$, wo der Funktionswert $f(0) = 0$ ist.
Es folgt die erste Aussage des Satzes.
Für die zweite Aussage schauen wir uns die Ränder des Intervalls an sowie die Stelle(n) y mit $f'(y) = 0$. Zunächst die Ränder: $f(0) = 0$ und $f(1) = 2 \cdot \ln 2 - 1 - c = 0$.
Für f' zeigen wir nun, dass es genau eine Nullstelle y im Intervall $[0, 1]$ gibt und dass an der Stelle y die zweite Ableitung negativ ist, also f dort ein lokales Maximum und kein lokales Minimum hat.
Es ist $f''(x) = \frac{1}{1+x} - 2c$. Diese Funktion ist stetig und monoton fallend auf $[0, 1]$. An den Rändern gilt $f''(0) = 1 - 2c > 0$ und $f''(1) = (1/2) - 2c < 0$. Es gibt also ein $x_0 \in (0, 1)$ mit $f''(x_0) = 0$, $f''(x) > 0$ für alle $x \in [0, x_0)$ und $f''(x) < 0$ für alle $x \in (x_0, 1]$. Also ist f' monoton wachsend auf dem Intervall $[0, x_0]$ und monoton fallend auf dem Intervall $[x_0, 1]$. Da $f'(1)$ negativ und $f'(0)$ positiv ist, hat f' wegen des skizzierten Verlaufs auf dem Intervall $(x_0, 1]$ genau eine Nullstelle y_0, an dieser ist $f''(y_0) < 0$ und die Funktion f hat an der Stelle y_0 ein lokales Maximum und kein lokales Minimum. □

Random Walks und elektrische Netzwerke

Hier wollen wir zeigen, dass das im entsprechenden Kapitel angegebene Gleichungssystem unter der Zusatzbedingung, dass eine der Variablen $pot_i = 0$ gesetzt wird, höchstens eine Lösung besitzt.
Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$. Das Gleichungssystem, das sich bei der Darstellung als elektrisches Netzwerk ergibt, lässt sich beschreiben als $A \cdot x = b$, wobei die $n \times n$-Matrix A wie folgt aussieht:
$A_{ii} = \text{grad}_i$ und für $i \neq j$ ist $A_{ij} = -1$, falls es eine Kante zwischen Knoten i und Knoten j gibt und $A_{ij} = 0$ sonst. Wir bemerken am Rande, dass die Matrix A symmetrisch ist. Wir zerlegen nun die Matrix A in Summanden:
Für eine Kante $e = \{i, j\}$ kann man die $n \times n$-Matrix Matrix J_e konstruieren, die überall Nullen enthält, außer: An Position (i, i) steht eine 1, an Position (j, j) steht eine 1, an Position (i, j) und (j, i) steht jeweils eine -1. Offensichtlich kann man die Matrix A schreiben als:

$$A = \sum_{e \in E} J_e.$$

Um zu zeigen, dass das Gleichungssystem höchstens eine Lösung hat, untersuchen wir, für welche x das Produkt $A \cdot x$ der Nullvektor ist.
Wenn $A \cdot x$ der Nullvektor ist, dann ist $x^T \cdot A \cdot x$ gleich Null.
Es ist

$$x^T \cdot J_e \cdot x = x_i^2 + x_j^2 - 2x_i x_j = (x_i - x_j)^2.$$

Dies ist immer größer oder gleich Null und auch nur dann Null, wenn $x_i = x_j$ ist.

$$\begin{aligned} x^T \cdot A \cdot x &= x^T \cdot (\sum_{e \in E} J_e) \cdot x \\ &= \sum_{e \in E} (x^T \cdot J_e \cdot x). \end{aligned}$$

Wenn $A \cdot x$ der Nullvektor sein soll, dann muss also $x^T \cdot A \cdot x$ gleich Null sein und somit für jede Kante $e = \{i, j\}$ im Graphen gelten, dass $x^T \cdot J_e \cdot x = 0$ und somit $x_i = x_j$ ist.
Wegen der Transitivität der Gleichheit folgt nun: Wenn der Graph G zusammenhängend ist, dann folgt aus $A \cdot x = 0$, dass $x_1 = x_2 = \cdots = x_n$ sein muss.
Wenn man nun die Zusatzbedingung stellt, dass eine der Variablen gleich 0 ist, dann erfüllt nur der Nullvektor die Bedingung $A \cdot x = 0$ und die Zusatzbedingung.
Nun folgt natürlich: Gäbe es zwei Vektoren x und x', die $A \cdot x = b$ und $A \cdot x' = b$ und die Zusatzbedingung erfüllen, dann würde $x - x' \neq 0$ die Bedingung $A \cdot (x - x') = 0$ und die Zusatzbedingung erfüllen. Also gibt es höchstens eine Lösung.