STOCHASTISCHE ALGORITHMEN

Wolfgang König

Vorlesungsskript

Universität zu Köln

Sommersemester 2003

Inhaltsverzeichnis

Vorwort

Dies ist das Vorlesungsskript für eine zweistündige Vorlesung über stochastische Algorithmen an der Universität zu Köln im Sommersemester 2003. Es werden für eine Reihe von Problemen randomisierte Lösungsalgorithmen vorgestellt, diskutiert und mit einander verglichen. Wir erläutern Vorzüge und Nachteile und fassen sie nach Möglichkeit mathematisch korrekt und beweisen sie. Insbesondere werden Aufgaben wie das approximative Zählen, das Sortieren und Vergleichen großer Mengen und das Ziehen einer Stichprobe mit einer gegebenen Verteilung behandelt, und es werden Algorithmen vom Las-Vegas-Typ diskutiert sowie Monte-Carlo-Methoden. Einen breiten Raum nehmen Markovkettenmethoden ein.

Diese Vorlesung lehnt sich an verschiedene Quellen an, und die Einflüsse der verschiedenen Texte sind manchmal (wenn sie nicht gesondert gekennzeichnet sind) nicht leicht von einander zu trennen. Die hauptsächlichen Quellen, aus denen geschöpft wurde, sind die Monographie [MR95] von Motwani und Raghavan über randomisierte Algorithmen, das im Werden befindliche Buch [AF03] über Markovketten auf Grafen und das Büchlein [H02] über algorithmische Anwendungen von Markovketten. Eine große Hilfe war auch das Skript [G00], dessen Zielrichtung sich mit dem des vorliegenden Skriptes stark überschneidet und dem etliche Anregungen entnommen wurden.

Die zu Grunde liegende Theorie, die wir voraus setzen, ist von geringem Umfang und umfasst im Wesentlichen nur die elementare Wahrscheinlichkeitstheorie, also die Theorie der Wahrscheinlichkeiten auf endlichen Mengen, sowie die Theorie der endlichen Markovketten.

Notationen und Konventionen

Die meisten Schreib- und Sprechweisen werden beim ersten Auftauchen erklärt. Hier werden nur ein paar wenige Konventionen aufgelistet:

Wenn in diesem Skript von 'zufällig' ohne weiteren Zusatz gesprochen wird, dann ist gemeint: 'nach der gleichförmigen Verteilung verteilt', d. h. jede Möglichkeit hat die selbe Wahrscheinlichkeit (falls es endlich viele sind). [1]

Wir werden keinen Gebrauch von Ganzzahlanteil-Klammern $\lfloor\cdot\rfloor$ oder $\lceil\cdot\rceil$ machen und bei Zahlen (wie etwa $n^{\frac{3}{4}}$), wenn sie eine Anzahl bezeichnen, still schweigend davon ausgehen, dass sie natürliche Zahlen sind.

Für Folgen $(a_n)_{n\in\mathbb{N}}$ und $(b_n)_{n\in\mathbb{N}}$ von positiven reellen Zahlen benutzen wir manchmal folgende Schreibweisen für asymptotische Vergleichsaussagen, jeweils für $n \to \infty$:

$$\begin{aligned}
a_n = \mathcal{O}(b_n) \qquad &\iff \qquad \left(\frac{a_n}{b_n}\right)_{n\in\mathbb{N}} \text{ ist beschränkt,}\\
a_n = \Omega(b_n) \qquad &\iff \qquad \left(\frac{b_n}{a_n}\right)_{n\in\mathbb{N}} \text{ ist beschränkt, also } b_n = \mathcal{O}(a_n),\\
a_n = \Theta(b_n) \qquad &\iff \qquad a_n = \mathcal{O}(b_n) \text{ und } b_n = \mathcal{O}(a_n),\\
a_n = o(b_n) \qquad &\iff \qquad \lim_{n\to\infty} \frac{a_n}{b_n} = 0.
\end{aligned}$$

[1] Dies wird von anderen Autoren oft mit 'ganz zufällig' oder 'völlig zufällig' ausgedrückt.

Kapitel 1

Einführung

In diesem vorbereitenden Kapitel führen wir in die Problematik ein, sprechen ein paar einfache Beispiele an und schneiden die Frage der Erzeugung von Zufall auf dem Computer an.

1.1 Motivation

In vielen Bereichen tauchen Aufgaben bzw. Probleme auf, die von endlicher, aber immenser Größe sind, wie zum Beispiel die folgenden.

- Sortieren einer langen Liste von Zahlen,
- Auffinden des k-t größten Elementes in einer langen Liste von Zahlen,
- Mischen eines großen Kartenstapels,
- Auffinden einer gewissen Zeichenfolge in einem langen Text,
- Prüfen zweier langer Zahlenreihen auf Gleichheit,
- Prüfen einer großen Zahl auf Primeigenschaft,
- Auffinden eines optimalen Wertes einer Funktion auf einer endlichen großen Menge,
- Auswertung der Partitionssumme oder des Erwartungswertes einer Größe in einem physikalischen Partikelsystem,
- Auszählung einer großen Menge,
- Zustellung von Paketen in einem großen Netzwerk von Adressen,
- Ziehen einer Stichprobe aus einer großen Menge mit einer gegebenen Verteilung.

Bei vielen solcher Probleme sind gewisse (deterministische) Lösungsalgorithmen augenfällig, mit denen man systematisch zum Ziel kommt, indem sämtliche möglichen Fälle, die auftreten können, behandelt werden. Es ist oft klar, dass die Ausführung dieses Algorithmus bei umfangreichen Eingaben eine gigantische Zeitdauer erfordert, die in einigen Fällen z. B. die bisherige

Lebensdauer der Erde bei weitem übersteigt. Außerdem erfordert das systematische Erfassen aller möglichen Fälle oft einen riesigen Speicherplatz.

In vielen solcher Fälle hat es sich als äußerst effektiv erwiesen, geeignete *randomisierte*, also *zufällige* Algorithmen zu entwickeln und einzusetzen. Der Vorteil eines zufälligen gegenüber einen deterministischen Algorithmus besteht oft in kürzerer Laufzeit oder in geringerem Speicherbedarf; er muss ja meist nicht *alle* Fälle erfassen und korrekt behandeln, sondern nur die 'typischen'. Die Kunst besteht darin, die Entscheidungen des Algorithmus so zu randomisieren, dass ungünstige Verläufe mit sehr geringer Wahrscheinlichkeit eintreten, bzw. dass das zufällige Erkunden nur eines kleinen Teils der zu untersuchenden Menge schon mit hoher Wahrscheinlichkeit sehr viel relevante Information erbringt.

Die Nachteile eines zufälligen Algorithmus bestehen oft darin, dass die Lösung entweder nicht mit absoluter Sicherheit gefunden wird oder nicht mit absoluter Präzision oder dass geeignete Abbruchkriterien nicht leicht zu geben sind. Bei Entscheidungsproblemen z. B. wird nur eine zufällige Auswahl von Tests durchgeführt, sodass man bei Nichtbestehen zwar eine definitive Antwort erhält, bei Bestehen aber nur eine hoch wahrscheinlich korrekte. Und viele auf dem Langzeitverhalten von Markovketten basierende Algorithmen erreichen in aller Regel nie die Gleichgewichtsverteilung der Kette, geben also nur eine approximative Lösung. Man nennt einen zufälligen Algorithmus, der im Falle der Termination stets eine korrekte Lösung ausgibt, einen *Las-Vegas-Algorithmus*, und einen Algorithmus, der mit positiver Wahrscheinlichkeit eine falsche Lösung liefert, einen *Monte-Carlo-Algorithmus.* Bei Entscheidungsproblemen (wo also die Antwort nur 'Ja' oder 'Nein' sein kann) unterscheidet man noch die Monte-Carlo-Algorithmen nach solchen mit *einseitigen Fehlern* bzw. *zweiseitigen Fehlern*, wobei letztere bei beiden möglichen Antworten mit positiver Wahrscheinlichkeit irren können, erstere aber nur bei einer der beiden, und die andere ist immer korrekt.

1.2 Ein simples Beispiel

Das folgende (recht künstliche) Beispiel demonstriert die Tatsache, dass zufällige Algorithmen schon in sehr simplen Situationen gegenüber deterministischen Algorithmen große Vorteile haben können.

Beispiel 1.2.1 (verrückter Millionär). Ein verrückter Millionär sendet an 20 vernunftbegabte Menschen (sagen wir, Stochastiker) je einen gleich lautenden Brief mit dem folgenden Inhalt: 'Diesen Brief haben auch gewisse 19 andere Stochastiker erhalten. Sie können nun wählen, ob Sie mir einen Brief schicken oder nicht. Wenn genau ein Brief von Ihnen 20 bei mir eintrifft, dann erhält der Absender eine Million Euro. Falls keiner oder mehr als einer eintreffen, dann erhält keiner von Ihnen etwas. Allerdings dürfen Sie unter einander keinen Kontakt aufnehmen.'

Man steht nun vor dem Problem: abschicken oder nicht, was ist die beste Strategie? Wir dürfen davon ausgehen, dass die anderen 19 Personen unabhängig von einander die selbe Strategie einschlagen, wenn sie denn die beste ist. Allerdings gibt es nur zwei deterministische Strategien: das Abschicken und das Nichtabschicken. Beide Strategien haben ja nach den Regeln des Millionärs die Erfolgsaussicht Null.

Andererseits (und das ist die Lösung) gibt es ja eine Menge randomisierter Strategien: Wir schicken mit Wahrscheinlichkeit p ab, wobei $p \in (0,1)$ ein Parameter ist! Die Erfolgsaussicht kann man leicht berechnen (immer unter der Voraussetzung, dass die anderen unabhängig der selben Strategie folgen): Mit Wahrscheinlichkeit $p(1-p)^{19}$ erhält man die Million. Dieser Wert wird maximal, wenn man $p = \frac{1}{20}$ wählt, wie man leicht heraus findet. Also bauen wir uns eine Apparatur, die mit Wahrscheinlichkeit $\frac{1}{20}$ entscheidet, und im positiven Fall schicken wir den Brief ab, sonst nicht.

Hier ist also ein simpler randomisierter Algorithmus jedem deterministischen weit überlegen. $\diamond$

1.3 Zufall auf dem Computer

Es sei noch kurz das Problem des Erzeugens von Zufalls auf dem Computer angesprochen.

Bei der Ausführung eines zufälligen Algorithmus auf dem Computer stellt sich unweigerlich das Problem nach der Bereitstellung von Zufallszahlen. Die meisten modernen Computer sind mit einem Generator von Zufallszahlen ausgestattet, und die Ergebnisse bestehen die meisten statistischen Tests auf Zufälligkeit. Man ist deshalb geneigt zu glauben, dass man immer z. B. auf eine Folge von unabhängigen, auf dem Intervall $[0, 1]$ gleichförmig verteilten Zufallszahlen zurück greifen kann. (Dies ist eine Forderung, die man z. B. für die Simulation einer Markovkette stellen muss.)

Tatsächlich aber ist dies falsch, und zwar aus mindestens zwei Gründen: (1) Der Erzeugung dieser ‘zufälligen’ Zahlen im Computer liegt im Normalfall eine komplizierte Kette von *deterministischen* Operationen zu Grunde, und das widerspricht jeder intuitiven Definition von Zufälligkeit. (2) Diese Zufallszahlen sind *endliche* Bit-Folgen, also rationale Zahlen, während man leicht zeigen kann, dass eine auf $[0, 1]$ gleichförmig verteilte Zahl mit Wahrscheinlichkeit Eins irrational ist. Besonders aus dem ersten Grund nennt man diese Zufallszahlen treffender *Pseudozufallszahlen*, denn sie haben nur die Eigenschaft, nicht oder schwer unterscheidbar zu sein von ‘echten’ Zufallszahlen, in dem Sinn, dass die meisten der bekannten statistischen Tests keinen signifikanten Unterschied feststellen können.

Bemerkung 1.3.1 (lineare Kongruenzmethode). Die Generierung von Pseudozufallszahlen ist recht schnell und in vielen Programmen standardmäßig eingebaut. Sie funktioniert oft nach dem folgenden Schema, der sogenannten *linearen Kongruenzmethode*: Man wählt ein ‘Modul’ m, etwa $m = 2^{32}$, einen Faktor a, etwa $a = 69069$, und ein Inkrement $b \in \mathbb{N}$, etwa $b = 1$. Dann wählt man willkürlich einen Startwert $k_0 \in \{0, \dots, m-1\}$ und setzt iterativ $k_{i+1} = (ak_i + b) \bmod m$ für $n \in \mathbb{N}$. Die gewünschte Folge von ‘unabhängigen auf $[0, 1]$ gleichförmig verteilten Zufallszahlen’ ist dann die Folge der $u_i = k_i/m$ mit $i \in \mathbb{N}$. Die Parameter a und b müssen mit Geschick gewählt werden, damit möglichst viele statistische Tests auf Zufälligkeit bestanden werden. In den 1960er Jahren war die oben beschriebene Methode mit $a = 65539$, $m = 2^{31}$ und $b = 0$ weit verbreitet, bis man heraus fand, dass es gewisse 15 Ebenen im $\mathbb{R}^3$ gibt, in denen je alle drei Tripel u_i, u_{i+1}, u_{i+2} von aufeinander folgenden dieser Zufallszahlen liegen. Lästige Eigenschaften dieser Struktur besitzen alle diese Algorithmen, aber durch geschickte Wahl der Parameter kann man diese Defekte gering halten. $\diamond$

Wir werden nun immer idealistischerweise davon ausgehen, dass eine Folge von unabhängigen, auf $[0, 1]$ gleichförmig verteilten Zufallszahlen $U_0, U_1, U_2, \dots$ zur Verfügung steht. Betrachten wir noch kurz das Problem, wie man dann eine beliebige vorgegebene Verteilung simulieren kann:

Beispiel 1.3.2 (Simulation einer beliebigen Verteilung). Es sei U eine gleichförmig auf $[0, 1]$ verteilte Zufallsgröße, und es sei Q eine beliebige Verteilung auf $\mathbb{R}$ mit Verteilungsfunktion

F. Wir definieren die Umkehrfunktion F^{-1} von F durch $F^{-1}(u) = \inf\{x \in \mathbb{R} \colon F(x) \geq u\}$. Dann hat die Zufallsvariable $F^{-1}(U)$ die Verteilungsfunktion F, also die Verteilung Q, wie man leicht als Übungsaufgabe verifiziert. Für etliche bekannte Verteilungen ist F^{-1} natürlich explizit bekannt. $\diamond$

Beispiel 1.3.3 (Simulation von Verteilungen auf endlichen Mengen). Eine beliebig vorgegebene Verteilung π auf einer endlichen Menge S kann mit Hilfe einer auf $[0, 1]$ gleichförmig verteilten Zufallsgröße U wie folgt simuliert werden: Wir fixieren eine stückweise stetige Funktion $\psi \colon [0, 1] \to S$, sodass für jedes $s \in S$ die Totallänge der Intervalle, auf denen ψ den Wert s annimmt, gleich $\pi(s)$ ist. Dies kann man z. B. bewerkstelligen, indem man $S = \{s_1, \dots, s_k\}$ numeriert und ψ auf dem Intervall $(\sum_{j=1}^{i-1} \pi(s_j), \sum_{j=1}^{i} \pi(s_j)]$ gleich s_i setzt für $i = 1, \dots, k$, und $\psi(0) = s_1$. Dann hat die Zufallsgröße $\psi(U)$ die Verteilung π. (Dies ist natürlich ein Spezialfall der Methode von Beispiel 1.3.2.) Natürlich wird dieses Vorgehen unpraktikabel, wenn die Menge S sehr groß ist, wie es in vielen Fällen vorkommt. Die *effiziente* Erzeugung einer nach π verteilten Zufallsgröße wird uns später noch intensiv beschäftigen. $\diamond$

Beispiel 1.3.4 (Verwerfungsmethode). Es sei Q eine beliebige Verteilung, und es sei B ein Ereignis mit $Q(B) > 0$. Wie generiert man eine Zufallsgröße Z^*, deren Verteilung die bedingte Verteilung $Q(\cdot \mid B)$ ist? Hierzu seien unabhängige Zufallsgrößen $Z_1, Z_2, \dots$ gegeben, die alle die Verteilung Q haben, also $\mathbb{P}(Z_i \in A) = Q(A)$ für alle Ereignise A. Man betrachtet den Index

$$\tau = \inf\{n \in \mathbb{N} \colon Z_n \in B\}$$

der ersten dieser Variablen, die in B liegt, und dann setzt man $Z^* = Z_\tau$. Durch eine Aufspaltung nach den Werten von τ kann man leicht verifizieren, dass Z^* die gewünschte Verteilung besitzt, denn für alle Ereignisse A gilt

$$\begin{aligned}
\mathbb{P}(Z^* \in A) &= \sum_{n \in \mathbb{N}} \mathbb{P}(Z_n \in A, \tau = n) \\
&= \sum_{n \in \mathbb{N}} \mathbb{P}(Z_1 \notin B, \dots, Z_{n-1} \notin B, Z_n \in A \cap B) \\
&= \sum_{n \in \mathbb{N}} (1 - Q(B))^{n-1} Q(A \cap B) = Q(A \mid B).
\end{aligned}$$

$\diamond$

'Echte' Zufälligkeit (was auch immer das sein soll) ist bisher noch nicht erreicht worden, sodass hier also durchaus eine Fehlerquelle bei der Implementierung zufälliger Algorithmen entstehen kann. Nach dem derzeitigen Stand vermutet man, dass eine Erzeugung von Zufallszahlen auf physikalischem Wege sehr langsam sein würde im Vergleich zu der Geschwindigkeit beim Ausführen arithmetischer Operationen. Daher wird manchmal bei der Ermittlung des Aufwandes eines randomisierten Algorithmus auch die Anzahl der benötigten 'echt zufälligen' Bits heran gezogen, aber das wollen wir nicht weiter verfolgen.

Auf das Problem der Erzeugung von Zufall im Computer ist eine Menge Arbeit verwendet worden, siehe etwa [K98, Chapter 3]. Eine umfangreiche Tabelle von zufälligen Zahlen erhält man unter `http://www.rand.org/publications/classics/randomdigits/`. Dort wird auch die Generierung beschrieben; im Wesentlichen wurde ein elektronisches Roulettespiel benutzt, die etwa eine Zahl pro Sekunde produzierte. Auf der Netzseite `http://www.fourmilab.ch/hotbits/` werden 'zufällig aussehende' Zahlen angeboten, die auf physikalischem Wege erzeugt werden. Ein Ansatz, der auf algorithmischer Komplexitätstheorie basiert, wird in [G99] vorgestellt.

Kapitel 2

Such- und Sortieralgorithmen

In diesem Kapitel stellen wir stochastische Algorithmen vor, die eine lange Liste von Zahlen sortieren bzw. das k-t größte Element identifizieren.

2.1 Randomisierter Quicksort

Dieser Algorithmus sortiert eine endliche Menge von Zahlen rekursiv durch sukzessives binäres zufälliges Zerlegen:

Randomisierter Quicksort

Eingabe: Eine Menge S von n paarweise verschiedenen Zahlen.

Ausgabe: Die Elemente von S in aufsteigender Ordnung.

Algorithmus:

1. Wähle zufällig ein Element Y aus S.
2. Bestimme die Mengen S_1 und S_2, die aus den Elementen von S bestehen, die kleiner bzw. größer als Y sind.
3. Gebe S_1, Y und S_2 in dieser Reihenfolge aus.
4. Wende Schritte 1 - 3 jeweils auf S_1 und S_2 rekursiv an, falls sie jeweils mehr als ein Element besitzen.

Dieser Algorithmus basiert auf dem in [H61b] eingeführten Algorithmus *Quicksort*.

Es ist bei Sortieralgorithmen üblich, die Gesamtzahl aller vom Algorithmus durchgeführten Vergleiche als Maß für den Aufwand zu wählen. Sei X_n diese Anzahl für den Randomisierten Quicksort. Bei günstigem Verlauf des Algorithmus (d. h. wenn immer zufälligerweise Y gleich dem Median von S ist, so dass die Mengen S_1 und S_2 etwa gleich groß sind), sind nur etwa $\log_2(n)$ Iterationen durchzuführen, die jeweils etwa n Vergleiche benötigen, also ist dann X_n nur etwa von der Größenordnung $n \log n$. Im ungünstigen Fall allerdings (d. h. wenn 'dummerweise' immer das kleinste oder größte Element von S als Y gewählt wird), braucht man etwa n Iterationen und je n Vergleiche, als ist X_n von der Ordnung n^2. Der große Vorteil des Randomisierten Quicksort ist allerdings, dass ungünstige Verläufe eine so geringe Wahrscheinlichkeit haben, dass X_n im

Durchschnitt von der Ordnung $n \log n$ ist:

Satz 2.1.1. *Für alle $n \in \mathbb{N}$ gilt $\mathbb{E}(X_n) = 2(n+1)\sum_{k=1}^n \frac{1}{k} - 4n$.*

Man beachte, dass $\sum_{k=1}^n \frac{1}{k} = \log n + \gamma + o(1)$ für $n \to \infty$, wobei $\gamma \approx 0,577$ die *Euler-Mascheroni*-Konstante heißt. Wir geben nun zwei Beweise von Satz 2.1.1.

Beweis A. Es sei $S = \{s_{(1)}, s_{(2)}, \dots, s_{(n)}\}$ die korrekte Ordnung von S, also $s_{(i)} < s_{(j)}$ für $i < j$, d. h. $s_{(i)}$ ist das i-t kleinste Element. Dann gilt

$$X_n = \sum_{1 \leq i < j \leq n} \mathbb{1}_{A_{ij}} \tag{2.1.1}$$

wobei A_{ij} das Ereignis bezeichnet, dass es im Verlauf des Algorithmus zum Vergleich von $s_{(i)}$ und $s_{(j)}$ kommt. Nun argumentieren wir, dass gilt:

$$\mathbb{P}(A_{ij}) = \frac{2}{j-i+1}, \qquad 1 \leq i < j \leq n. \tag{2.1.2}$$

Dies basiert auf den folgenden zwei Tatsachen:

1. Das Ereignis A_{ij} tritt genau dann ein, wenn entweder $s_{(i)}$ oder $s_{(j)}$ das erste der Elemente $s_{(i)}, \dots, s_{(j)}$ ist, das als Vergleichselement Y heran gezogen wird. Denn falls ein anderes zuerst gewählt würde, sagen wir $s_{(k)}$ mit $i < k < j$, dann würden $s_{(i)}$ und $s_{(j)}$ zwar je mit $s_{(k)}$ verglichen werden, aber nicht mit einander, denn sie gehören dann zu verschiedenen Teilmengen S_1 (links von $s_{(k)}$) und S_2 (rechts von $s_{(k)}$), die ja separat behandelt werden.

2. Jedes der Elemente $s_{(i)}, \dots, s_{(j)}$ hat die selbe Wahrscheinlichkeit, als erstes als Vergleichselement gewählt zu werden.

Also ist $\mathbb{P}(A_{ij})$ die Wahrscheinlichkeit, dass $s_{(i)}$ oder $s_{(j)}$ aus den $j-i+1$ gleich wahrscheinlichen Elementen $s_{(i)}, \dots, s_{(j)}$ zu erst gezogen werden, also gleich $\frac{2}{j-i+1}$, d. h. (2.1.2) ist bewiesen. Setzen wir (2.1.2) in (2.1.1) ein, so ergibt sich

$$\mathbb{E}(X_n) = 2 \sum_{1 \leq i < j \leq n} \frac{1}{j-i+1}. \tag{2.1.3}$$

Nun können wir die angegebene Formel für $\mathbb{E}(X_n)$ leicht mit einer Vollständigen Induktion verifizieren, denn man rechnet leicht nach, dass gilt:

$$\begin{gathered}\mathbb{E}(X_{n+1}) = \mathbb{E}(X_n) + 2(H_{n+1} - 1), \\ 2(n+2)H_{n+1} - 4(n+1) = 2(n+1)H_n - 4n + 2(H_{n+1} - 1),\end{gathered}$$

wobei wir $H_n = \sum_{k=1}^n \frac{1}{k}$ gesetzt haben. □

Beweis B. Die Verteilung von X_n erfüllt die Rekursionsgleichung

$$X_n \stackrel{\mathcal{D}}{=} n - 1 + X^{(1)}_{U_n - 1} + X^{(2)}_{n-U_n}, \qquad n \in \mathbb{N} \setminus \{1\}, \tag{2.1.4}$$

wobei $X_k^{(1)}$ und $X_k^{(2)}$ unabhängige Zufallsgrößen mit der Verteilung von X_k sind, und U_n ist eine auf $\{1, \dots, n\}$ gleichförmig verteilte Zufallsgröße, die unabhängig von den $X_k^{(i)}$ ist. (Wir schreiben $\stackrel{\mathcal{D}}{=}$ für Gleichheit in Verteilung.) Hierbei steht $n-1$ für die Anzahl der Vergleiche mit dem im ersten Schritt ausgewählten Element $Y \stackrel{\mathcal{D}}{=} U_n$, und im weiteren Verlauf wird die Menge S_1, die ja $U_n - 1$ Elemente hat, mit $X^{(1)}_{U_n-1}$ Vergleichen geordnet, und die $(n-U_n)$-elementige Menge S_2 mit $X^{(2)}_{n-U_n}$ Vergleichen.

Wir bilden nun den Erwartungswert in (2.1.4), spalten nach den Werten von U_n auf und benutzen den Satz von der totalen Wahrscheinlichkeit, um für $n \geq 2$ zu erhalten:

$$\begin{aligned}
\mathbb{E}(X_n) &= n - 1 + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}(X^{(1)}_{U_n-1} \mid U_n = k) + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}(X^{(2)}_{n-U_n} \mid U_n = k) \\
&= n - 1 + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}(X_{k-1}) + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}(X_{n-k}) \\
&= n - 1 + \frac{2}{n}\sum_{k=0}^{n-1} \mathbb{E}(X_k).
\end{aligned}$$

Daraus folgt leicht, dass $\mathbb{E}(X_n)$ die Rekursionsgleichung

$$\mathbb{E}(X_{n+1}) = \frac{n+2}{n+1}\mathbb{E}(X_n) + \frac{2n}{n+1}, \qquad n \in \mathbb{N},$$

mit Startwert $\mathbb{E}(X_1) = 0$ erfüllt. Man sieht leicht, dass auch die rechte Seite der behaupteten Formel diese Rekursionsgleichung erfüllt. $\square$

Bemerkung 2.1.2. (a) Die Verteilung von X_n hängt nicht von S ab.

(b) Der (deterministische) Sortieralgorithmus *Heapsort* benötigt zwar ebenfalls $\mathcal{O}(n \log n)$ Vergleiche, aber der Vorfaktor ist doppelt so groß wie beim Randomisierten Quicksort (außerdem ist Heapsort komplizierter zu implementieren).

(c) Satz 2.1.1 betrifft nur den *erwarteten* Aufwand des Algorithmus. Eine noch bessere Aussage über die Qualität wäre aber eine Aussage, dass X_n mit hoher Wahrscheinlichkeit in der Nähe seines Erwartungwertes liegt. Eine solche Aussage wird in [R91] bewiesen: Für beliebige $\lambda, \varepsilon > 0$ existiert eine Konstante $c(\lambda, \varepsilon) > 0$, sodass

$$\mathbb{P}\big(X_n > (1+\varepsilon)\mathbb{E}(X_n)\big) \leq c(\lambda, \varepsilon) n^{-2\lambda\varepsilon}, \qquad n \in \mathbb{N}.$$

2.2 Find

Ein eng mit dem Randomisierten Quicksort verwandter Algorithmus ist der Algorithmus *Find*, der in einer gegebenen Menge S mit n Elementen das k-t kleinste heraus findet. Dazu führt Find die ersten zwei Schritte des Randomisierten Quicksort durch, d. h. er zerlegt S in S_1, $\{Y\}$ und S_2, sodass alle Elemente in S_1 kleiner als ein zufällig gewähltes Element Y von S ist und alle Elemente in S_2 größer sind. Falls $|S_1| = k-1$, so ist Y das gesuchte Element, und Find terminiert. Ansonsten bestimmt Find rekursiv das Element vom Rang k in S_1, wenn $|S_1| \geq k$, und das Element vom Rang $k - |S_1| - 1$ in S_2, falls $|S_1| < k-1$.

Sei $X_{n,k}$ die Anzahl der Vergleiche, die Find benötigt. Es stellt sich heraus, dass $X_{n,k}$ im Durchschnitt nur linear in n wächst, gleichmäßig in k:

Satz 2.2.1. *Für jedes $n \in \mathbb{N}$ und jedes $k \in \{1, \dots, n\}$ gilt $\mathbb{E}(X_{n,k}) \leq 4n$.*

Beweisskizze. Wie in Beweis B von Satz 2.1.1 benutzt man die rekursive Struktur des Algorithmus, um zu sehen, dass die Rekursionsgleichung

$$\mathbb{E}(X_{n,k}) = n - 1 + \frac{1}{n} \sum_{l=0}^{k-2} \mathbb{E}(X_{n-l-1,k-l-1}) + \frac{1}{n} \sum_{l=k}^{n-1} \mathbb{E}(X_{l,k}), \qquad n \geq 2,$$

für alle $k \in \{1, \dots, n\}$ gilt. Nun führt eine direkte Induktion über n zum Ziel. □

2.3 Lazy Select

Wie der Algorithmus Find bestimmt auch der Algorithmus *Lazy Select* (eingeführt in [FR75]) das Element $S_{(k)}$ vom Rang k in einer endlichen Menge S von paarweise verschiedenen Zahlen. Die folgende Notation ist hilfreich: Falls $S = \{s_1, s_2, \dots, s_n\}$ mit $s_1 < s_2 < \dots < s_n$, so nennen wir i den *Rang* von s_i in S und schreiben $i = \text{rang}_S(s_i)$ und $s_i = S_{(i)}$.

Die Strategie ist die folgende. Wir ziehen eine Stichprobe R vom Umfang r (später wählen wir $r = n^{\frac{3}{4}}$). Der Rang von $S_{(k)}$ in R wird ungefähr $x = k\frac{r}{n}$ sein. Daher sollte das Intervall $I = [R_{(x-m)}, R_{(x+m)}]$ (wobei wir später $m = \sqrt{n}$ wählen) mit hoher Wahrscheinlichkeit das gesuchte $S_{(k)}$ enthalten. Falls dies stimmt und falls $|I|$ nicht zu groß ist (ansonsten versuchen wir es noch einmal mit einer anderen Stichprobe), dann bestimmen wir den Rang von $S_{(k)}$ in I, und daraus können wir mit Hilfe des Ranges von $R_{(x-m)}$ in S den Rang von $S_{(k)}$ in S bestimmen.

Die Idee ist, dass nur die relativ kleinen Mengen R und I sortiert werden müssen, was den Aufwand gering hält. Eine formalere Beschreibung des Algorithmus ist wie folgt. Wir setzen die Parameter dabei ein als $r = n^{\frac{3}{4}}$ und $m = \sqrt{n}$.

Lazy Select:
Eingabe: Eine Menge S von n paarweise verschiedenen Zahlen und ein $k \in \{1, \dots, n\}$.
Ausgabe: Das Element $S_{(k)}$ vom Rang k in S.
Algorithmus:

1. Ziehe mit Zurücklegen eine Stichprobe R der Größe $n^{\frac{3}{4}}$ aus S.
2. Sortiere R mit einem guten Algorithmus. Setze $x = kn^{-\frac{1}{4}}$, $\ell = \max\{x - \sqrt{n}, 1\}$ und $h = \min\{x + \sqrt{n}, n\}$.
3. Bestimme die Ränge von $R_{(\ell)}$ und $R_{(h)}$ in S durch Vergleich mit allen Elementen aus S. Setze
$$Q = \begin{cases} S \cap (-\infty, R_{(h)}], & \text{falls } k < n^{\frac{1}{4}} \log n, \\ S \cap [R_{(\ell)}, R_{(h)}], & \text{falls } k \in [n^{\frac{1}{4}} \log n, n - n^{\frac{1}{4}} \log n], \\ S \cap [R_{(\ell)}, \infty), & \text{falls } k > n - n^{\frac{1}{4}} \log n. \end{cases} \tag{2.3.1}$$
Überprüfe, ob $S_{(k)} \in Q$ gilt, durch Vergleich von k mit den Rängen von $R_{(\ell)}$ und $R_{(h)}$ in S, und ob $|Q| \leq 4n^{\frac{3}{4}} + 2$. Falls das Ergebnis negativ ist, wiederhole Schritte 1 bis 3.
4. Sortiere Q. Identifiziere das Element $S_{(k)}$ als das Element in Q mit Rang $k - \text{rang}_S(R_{(\ell)}) + 1$, falls $k \geq n^{\frac{1}{4}} \log n$ bzw. als jenes mit Rang k, falls $k < n^{\frac{1}{4}} \log n$.

Für diese Wahl der Parameter können wir mit relativ geringem Aufwand das folgende (nicht optimale) Resultat beweisen:

Satz 2.3.1. *Der Algorithmus Lazy Select findet das Element $S_{(k)}$ mit Wahrscheinlichkeit $1 - \mathcal{O}(n^{-\frac{1}{4}})$ bei einmaligem Durchlauf der Schritte 1 bis 3. Er führt dabei maximal $2n + o(n)$ Vergleiche durch.*

Beweis. Für das Sortieren der Mengen R und Q in den Schritten 2 und 4 benötigt ein guter Sortieralgorithmus (wie etwa der *Heapsort*) $\mathcal{O}(n^{\frac{3}{4}} \log n)$ Vergleiche, da beide Mengen $\mathcal{O}(n^{\frac{3}{4}})$ Elemente besitzen. Die Kosten von Lazy Select werden also durch die $2(n-2)$ Vergleiche in Schritt 3 dominiert. Damit ist die zweite Aussage bewiesen.

Betrachten wir nun das Ereignis, dass Lazy Select das Element $S_{(k)}$ nicht bei einmaliger Durchführung der Schritte 1 bis 3 findet. Wir beschränken uns dabei auf den Fall $k \in [n^{\frac{1}{4}} \log n, n - n^{\frac{1}{4}} \log n]$; die anderen Fälle behandelt man analog. Nach Definition gibt es zwei Möglichkeiten des Scheiterns: $S_{(k)}$ liegt nicht in Q, oder Q ist zu groß. Wir schätzen zunächst die Wahrscheinlichkeit für $S_{(k)} \notin Q$ ab, also für $S_{(k)} < R_{(\ell)}$ oder $S_{(k)} > R_{(h)}$. Es reicht, $\mathbb{P}(S_{(k)} < R_{(\ell)}) \leq \mathcal{O}(n^{-\frac{1}{4}})$ zu zeigen, denn der Beweis von $\mathbb{P}(S_{(k)} > R_{(h)}) \leq \mathcal{O}(n^{-\frac{1}{4}})$ läuft analog.

Es seien $r = n^{\frac{3}{4}}$ die Anzahl der Ziehungen für die Menge R und $R_1, R_2, \ldots, R_r$ die einzelnen Ziehungen dieser Stichprobe. Wir betrachten die Zählvariable

$$Y_n = \sum_{i=1}^{r} \mathbb{1}_{\{R_i \leq S_{(k)}\}},$$

die die Anzahl der Ziehungen angibt, die nicht über $S_{(k)}$ liegen. Dann ist nämlich das betrachtete Ereignis $\{S_{(k)} < R_{(\ell)}\}$ gleich dem Ereignis $\{Y_n \leq \ell - 1\}$, dass weniger als ℓ Ziehungen über $S_{(k)}$ liegen. Die Wahrscheinlichkeit von $\{Y_n \leq \ell - 1\}$ läßt sich elementar behandeln, da Y_n eine zu den Parametern r und k/n binomialverteilte Zufallsgröße ist. Insbesondere gelten

$$\mathbb{E}(Y_n) = r\frac{k}{n} = x \qquad \text{und} \qquad \mathbb{V}(Y_n) = r\frac{k}{n}\Big(1 - \frac{k}{n}\Big) = n^{\frac{3}{4}}\frac{k}{n}\Big(1 - \frac{k}{n}\Big) \leq \frac{1}{4}n^{\frac{3}{4}}.$$

In dem Fall, dass $\ell = x - \sqrt{n}$, folgt mit Hilfe der Tschebyschev-Ungleichung, dass

$$\mathbb{P}(S_{(k)} < R_{(\ell)}) = \mathbb{P}(Y_n < \ell) \leq \mathbb{P}\big(|Y_n - \mathbb{E}(Y_n)| > \sqrt{n}\big) \leq \frac{1}{n}\mathbb{V}(Y_n) \leq \frac{1}{4}n^{-\frac{1}{4}}.$$

Im anderen Fall, wo $\ell = 1$, gilt wegen $k \geq n^{\frac{1}{4}} \log n$:

$$\mathbb{P}(S_{(k)} < R_{(\ell)}) = \mathbb{P}(Y_n = 0) = \Big(1 - \frac{k}{n}\Big)^{n^{\frac{3}{4}}} \leq \big(1 - n^{-\frac{3}{4}} \log n\big)^{n^{\frac{3}{4}}} \leq \exp(-\log n) = \frac{1}{n}.$$

Dies zeigt, dass die Wahrscheinlichkeit der ersten Möglichkeit zu scheitern gleich $\mathcal{O}(n^{-\frac{1}{4}})$ ist. Nun müssen wir noch zeigen, dass $\mathbb{P}(|Q| > 4n^{\frac{3}{4}} + 2) \leq \mathcal{O}(n^{-\frac{1}{4}})$. Für alle i und j ist evident, dass gilt:

$$\{|Q| > i + 2\} = \{|S \cap [R_{(\ell)}, R_{(h)}]| > i + 2\} \subset \{R_{(\ell)} \leq S_{(j)}\} \cup \{R_{(h)} \geq S_{(j+i+2)}\}.$$

Dies wenden wir auf $i = 4n^{\frac{3}{4}}$ und $j = k - 2n^{\frac{3}{4}}$ an und erhalten

$$\mathbb{P}(|Q| > 4n^{\frac{3}{4}} + 2) \leq \mathbb{P}(R_{(\ell)} \leq S_{(k_\ell)}) + \mathbb{P}(R_{(h)} \geq S_{(k_h)}),$$

wobei $k_\ell = \max\{k - 2n^{\frac{3}{4}}, 1\}$ und $k_h = \min\{k + 2n^{\frac{3}{4}}, n\}$. Nun zeigt man analog zu dem obigen Teil des Beweises, dass die beiden Terme auf der rechten Seite höchstens $\mathcal{O}(n^{-\frac{1}{4}})$ sind. (Dabei übernehmen k_ℓ bzw. k_h die Rolle von k.) □

Bemerkung 2.3.2. (a) Da wir die Stichprobe R *mit* Zurücklegen ziehen, muss nicht $|R| = n^{\frac{3}{4}}$ gelten, aber Abweichungen geschehen mit geringer Wahrscheinlichkeit, wie man elementar berechnen kann. Das Ziehen mit Zurücklegen läßt sich leichter implementieren (man muss sich nicht die bereits gezogenen Elemente merken) und vereinfacht den Beweis von Satz 2.3.1.

(b) Die in Satz 2.3.1 angegebene Schranke für die Wahrscheinlichkeit des Scheiterns ist nicht optimal, da die Tschebyschev-Ungleichung nicht optimal ist. Man kann mit feineren Methoden (siehe die Theorie der Großen Abweichungen oder die Bernstein-Chernov-Schranke) exponentiellen Abfall der Wahrscheinlichkeiten $\mathbb{P}(S_{(k)} < R_{(\ell)})$ herleiten.

(c) Will man die Wahrscheinlichkeit des Scheiterns durch andere Wahlen der Parameter $r = |R|$ und m verringern, so erhöht dies die Zahl der Vergleiche, also vergrößert den $o(n)$-Term.

(d) Die besten derzeit bekannten deterministischen Algorithmen benötigen für allgemeines k im schlechtesten Fall $3n$ Vergleiche, und ihre Implementierung ist recht kompliziert. Im Fall des Medians, also $k = n/2$, kennt man Algorithmen, die mit $2n$ Vergleichen auskommen.

(e) Es gibt simple Verbesserungsmöglichkeiten. Offensichtlich braucht in Schritt 3 ein Element, das kleiner als $R_{(\ell)}$ bzw. größer als $R_{(h)}$ ist, nicht mehr mit $R_{(h)}$ bzw. $R_{(\ell)}$ verglichen zu werden. Wenn $k \leq \frac{n}{2}$, dann empfiehlt es sich, die Elemente aus S zunächst mit $R_{(\ell)}$ zu vergleichen, und mit $R_{(h)}$, wenn $k > \frac{n}{2}$. Die erwartete Anzahl der von Lazy Select durchgeführten Vergleiche ist dann durch $\frac{3}{2}n + o(n)$ beschränkt. Wenn k bzw. $n - k$ von kleinerer Ordnung als n ist, so ist diese erwartete Anzahl sogar nur $n + o(n)$, also um den Faktor 2 besser als der beste bekannte deterministische Algorithmus.

Kapitel 3

Beschleunigen von Las-Vegas-Algorithmen

(Dieses Kapitel folgt [LSZ93].)

Es sei X die Laufzeit eines Las-Vegas-Algorithmus, also eines Algorithmus, der im Falle der Terminierung eine korrekte Lösung ausgibt. Daher ist X eine Zufallsgröße mit Werten in $\mathbb{N} \cup \{\infty\}$, und der Wert ∞ kann mit positiver Wahrscheinlichkeit angenommen werden, wenn etwa der Algorithmus in einer Endlosschleife landet. In diesem Fall und auch wenn die Geduld aus irgend einem Grund begrenzt ist, wird man auf die Idee verfallen, den Algorithmus abzubrechen und unabhängig neu zu starten. Eventuell empfiehlt sich sogar, den Abbruch und Neustart jeweils nach einer fest gewählten Anzahl von Schritten zu iterieren. In diesem Kapitel werden Kriterien gegeben, nach wie vielen Schritten man dies tun sollte, um eine möglichst kurze erwartete Laufzeit dieses modifizierten Algorithmus zu erreichen.

Bei Qualitätskriterien für Las-Vegas-Algorithmen findet man oft nur eine Angabe der Form '$\mathbb{P}(X > k) < \frac{1}{e}$ für $k > \dots$', d. h. man macht sich nicht die Mühe, die Wahrscheinlichkeit für lange Wartezeiten unter *beliebig* kleine Schranken zu drücken. Der Grund, dass eine solche Angabe für praktische Zwecke ausreicht, ist die hier diskutierte Technik des Abbruchs und Neustarts.

In Abschnitt 3.1 diskutieren wir konstante Strategien und setzen voraus, dass wir die Verteilung von X kennen. In Abschnitt 3.2 werden allgemeinere Strategien betrachtet, und in Abschnitt 3.3 gehen wir davon aus, dass die Verteilung von X unbekannt ist.

3.1 Konstante Abbruchzeit

Betrachten wir zunächst den Fall, dass jeweils nach einer festen Anzahl von $k \in \mathbb{N}$ Schritten abgebrochen und unabhängig neu gestartet wird. Die Laufzeiten der einzelnen Läufe sind dann unabhängige Kopien $X_1, X_2, X_3, \dots$ der Zufallsgröße X, die jeweils beim Wert k abgeschnitten werden, bis die erste von ihnen einen Wert in $\{1, \dots, k\}$ annimmt, bis also der modifizierte Algorithmus mit dem korrekten Ergebnis terminiert. Die Gesamtlänge Y_k des Algorithmus ist also gegeben als

$$Y_k = k(N_k - 1) + X_{N_k}, \qquad \text{wobei } N_k = \inf\{i \in \mathbb{N} \colon X_i \leq k\}. \tag{3.1.1}$$

Hierbei ist N_k die Anzahl der Neustarts, bis im N_k-ten Versuch endlich terminiert wird. Die folgenden Tatsachen macht man sich leicht klar:

(a) N_k ist eine zum Parameter $\mathbb{P}(X > k)$ auf $\mathbb{N}$ geometrisch verteilte Zufallsgröße,

(b) X_{N_k} hat die bedingte Verteilung von X gegeben $\{X \leq k\}$,

(c) N_k und X_{N_k} sind unabhängig.

Im Folgenden ist die (nahezu triviale) Unterscheidung wichtig, ob k größer oder kleiner als $\min \operatorname{supp}(X) = \min\{x \in \mathbb{N} : \mathbb{P}(X = x) > 0\}$ ist, also die kleinste Laufzeit, die positive Wahrscheinlichkeit hat. Der letztere Fall ist trivial, denn wir haben dann $Y_k = \infty$. Wir werden auch immer den Wert $k = \infty$ zulassen; in diesem Fall haben wir $Y_\infty = X$ und $N_\infty = 1$. Halten wir zunächst eine Formel für den Erwartungswert von Y_k fest:

Lemma 3.1.1 (Erwartungswert von $\mathbf{Y_k}$). *Für jedes $k \in \mathbb{N} \cup \{\infty\}$ gilt*

$$\mathbb{E}(Y_k) = \begin{cases} \frac{\mathbb{E}(X \wedge k)}{\mathbb{P}(X \leq k)}, & \textit{falls } k \geq \min \operatorname{supp}(X), \\ \infty, & \textit{falls } 1 \leq k < \min \operatorname{supp}(X). \end{cases}$$

Ferner gilt $\lim_{k\to\infty} \mathbb{E}(Y_k) = \mathbb{E}(X) = \mathbb{E}(Y_\infty)$.

Beweis. Der triviale Fall, wo $k < \min \operatorname{supp}(X)$, ist klar, denn hier gilt $Y_k = \infty$ mit Wahrscheinlichkeit Eins. Der Fall $k = \infty$ ist ebenfalls klar, denn $Y_\infty = X$.

Kommen wir zum Fall $\min \operatorname{supp}(X) \leq k < \infty$. Hier benutzen wir (3.1.1) und die Bemerkungen danach, um zu errechnen:

$$\begin{aligned} \mathbb{E}(Y_k) &= k\mathbb{E}(N_k - 1) + \mathbb{E}(X \mid X \leq k) = k\frac{\mathbb{P}(X > k)}{\mathbb{P}(X \leq k)} + \frac{\mathbb{E}(X \mathbb{1}\{X \leq k\})}{\mathbb{P}(X \leq k)} \\ &= \frac{\mathbb{E}(k\mathbb{1}\{X > k\} + X\mathbb{1}\{X \leq k\})}{\mathbb{P}(X \leq k)} = \frac{\mathbb{E}(X \wedge k)}{\mathbb{P}(X \leq k)}. \end{aligned}$$

Die Zusatzaussage ergibt sich mit dem Satz von der monotonen Konvergenz wie folgt:

$$\lim_{k\to\infty} \mathbb{E}(Y_k) = \lim_{k\to\infty} \frac{\mathbb{E}(X \wedge k)}{\mathbb{P}(X \leq k)} = \frac{\mathbb{E}(X)}{\mathbb{P}(X < \infty)} = \mathbb{E}(X) = \mathbb{E}(Y_\infty),$$

wobei man im Fall $\mathbb{P}(X < \infty) < 1$ beachten muss, dass dann $\mathbb{E}(X) = \infty$ gilt. □

Im Folgenden werden wir ein k *optimal* nennen, wenn k den Wert von $\mathbb{E}(Y_k)$ minimiert. Wir kürzen ab:

$$\alpha(k) = \frac{\mathbb{E}(X \wedge k)}{\mathbb{P}(X \leq k)}, \qquad \min \operatorname{supp}(X) \leq k \leq \infty,$$

Mit

$$A = \{k \in \mathbb{N} \cup \{\infty\} : \alpha(k) = \alpha^*\}, \qquad \text{wobei } \alpha^* = \min_{k \in \mathbb{N} \cup \{\infty\}} \alpha(k),$$

bezeichnen wir die Menge der optimalen k nach Lemma 3.1.1. Auf Grund der Zusatzaussage in Lemma 3.1.1 ist A nicht leer. Als Übungsaufgabe zeige man, dass in dem Fall, wo X geometrisch verteilt ist, $A = \mathbb{N}_0 \cup \{\infty\}$ ist, die Abbildung $k \mapsto \alpha(k)$ also konstant. Als ganz grobe Faustregel kann man sagen, dass bei 'dicken' Schwänzen der Verteilung von X (d. h. wenn $\mathbb{P}(X > k)$ asymptotisch recht groß ist, etwa im Fall $\mathbb{E}(X) = \infty$), die Folge $\alpha(k)$ schnell groß werden sollte, also das Minimum bei kleinen k zu suchen sein sollte. Mit anderen Worten, bei dicken Schwänzen sollte es vorteilhaft sein, nach relativ wenigen Schritten den Abbruch und Neustart durchzuführen. Bei dünnen hingegen (also wenn $\mathbb{P}(X > k)$ sehr schnell gegen Null konvergiert), sollte ein Abbruch sich kaum lohnen.

Somit haben wir also das Problem des optimalen Abbruchs und Neustarts gelöst unter der Voraussetzung von konstanter Abbruchdauer. Das optimale oder die optimalen k hängen von der ganzen Verteilung von X ab, und ihre Kenntnis setzt die Kenntnis der Verteilung von X voraus. Da diese Verteilung stark abhängt von den Eingabewerten und da Informationen über die Verteilung von X sowieso nur sehr schwer zu erhalten sind, hat diese Lösung vorerst nur theoretischen Wert.

3.2 Variable Abbruchzeiten

Auf der Suche nach noch besseren Beschleunigungsverfahren könnte man versuchen, den ersten Abbruch und Neustart nach k_1 Schritten durchzuführen, den zweiten nach weiteren k_2 Schritten und so weiter. Nun hat man eine Folge $K = (k_i)_{i\in\mathbb{N}}$ von Abbruchzeiten und sucht nach einer optimalen Wahl von K. Hierbei nennen wir eine Folge $K = (k_i)_{i\in\mathbb{N}}$ optimal, wenn $\mathbb{E}(Y_K)$ minimal ist, wobei Y_K die Laufzeit des modifizierten Algorithmus mit Abbruchzeiten $k_1, k_2, k_3, \dots$ ist. Wir nennen K auch eine *Strategie*. Formal können wir schreiben

$$Y_K = k_1 + k_2 + \dots + k_{N-1} + X_N, \qquad N = \inf\{i \in \mathbb{N} \colon X_i \le k_i\}, \tag{3.2.1}$$

wobei wir uns erinnern, dass $(X_i)_{i\in\mathbb{N}}$ eine Folge von unabhängigen, wie X verteilten Zufallsgrößen ist.

Der folgende Satz zeigt insbesondere, dass konstante Strategien, wie wir sie im vorigen Abschnitt betrachtet haben, optimal sind.

Satz 3.2.1. *Falls $k_i \in A$ für alle $i \in \mathbb{N}$, so ist $(k_i)_{i\in\mathbb{N}}$ eine optimale Strategie.*

Beweis. Wir dürfen voraussetzen, dass $k_i \ge \min \operatorname{supp}(X)$ für jedes i, anderenfalls beseitigen wir alle k_i mit $k_i < \min \operatorname{supp}(X)$ und erhalten offensichtlich eine nicht schlechtere Strategie.

Wir werden $\mathbb{E}(Y_K)$ als eine Konvexkombination der $\alpha(k_1), \alpha(k_2), \dots$ schreiben, woraus sofort die Aussage folgt. Hierzu betrachten wir

$$p_j = \mathbb{P}(N = j) = \mathbb{P}(X \le k_j) \prod_{i=1}^{j-1} \mathbb{P}(X > k_i), \qquad j \in \mathbb{N},$$

und bemerken, dass $\sum_{j\in\mathbb{N}} p_j = 1$ ist. Wir werden zeigen, dass gilt:

$$\mathbb{E}(Y_K) = \sum_{j\in\mathbb{N}} p_j \alpha(k_j), \tag{3.2.2}$$

und dies wird den Beweis beenden. Wir kürzen ab: $n_i = k_1 + k_2 + \dots + k_i$, insbesondere $n_0 = 0$.

Wir zerlegen nach der Anzahl der Durchläufe mit Hilfe des Satzes von der totalen Wahrscheinlichkeit:

$$\mathbb{E}(Y_K) = \sum_{j=1}^{\infty} \mathbb{P}(N = j)\mathbb{E}(Y_K \mid N = j) = \sum_{j=1}^{\infty} p_j \sum_{\ell=1}^{k_j} (n_{j-1} + \ell)\mathbb{P}(Y_K = n_{j-1} + \ell \mid n_{j-1} < Y_K \le n_j).$$

Da im Fall $n_{j-1} < Y_K \leq n_j$ die Variable $Y_K - n_{j-1}$ die bedingte Verteilung von X gegeben $\{X \leq k_j\}$ hat (siehe (3.2.1)), können wir fortsetzen mit

$$\mathbb{E}(Y_K) = \sum_{j=1}^{\infty} p_j \big(n_{j-1} + \mathbb{E}(X \mid X \leq k_j)\big) = \sum_{j=1}^{\infty} p_j \Big(n_{j-1} + \alpha(k_j) - k_j \frac{\mathbb{P}(X > k_j)}{\mathbb{P}(X \leq k_j)}\Big)$$

nach Definition von $\alpha(k_j)$. Daraus folgt

$$\begin{aligned}
\mathbb{E}(Y_K) - \sum_{j \in \mathbb{N}} p_j \alpha(k_j) &= \sum_{j=1}^{\infty} p_j \Big(n_{j-1} - k_j \frac{\mathbb{P}(X > k_j)}{\mathbb{P}(X \leq k_j)}\Big) \\
&= \sum_{j=1}^{\infty} \Big(\frac{p_j n_{j-1}}{\mathbb{P}(X \leq k_j)} - n_{j-1} p_j \frac{\mathbb{P}(X > k_j)}{\mathbb{P}(X \leq k_j)} - k_j \frac{p_{j+1}}{\mathbb{P}(X \leq k_{j+1})}\Big) \\
&= \lim_{m \to \infty} \sum_{j=1}^{m} \Big(\frac{p_j n_{j-1}}{\mathbb{P}(X \leq k_j)} - n_j \frac{p_{j+1}}{\mathbb{P}(X \leq k_{j+1})}\Big) \\
&= - \lim_{m \to \infty} \frac{p_{m+1} n_m}{\mathbb{P}(X \leq k_{m+1})}.
\end{aligned}$$

Da $k_{m+1} \geq \min \operatorname{supp}(X)$, ist $\mathbb{P}(X \leq k_{m+1}) \geq \mathbb{P}(X = \min \operatorname{supp}(X)) > 0$, also ist nur zu zeigen, dass $\lim_{m\to\infty} p_m n_{m-1} = 0$. Wenn $\mathbb{E}(Y_K) < \infty$, so gilt $p_m n_{m-1} = (k_1 + \cdots + k_{m-1}) \mathbb{P}(N = m) \leq \mathbb{E}(Y_K \mathbb{1}\{N = m\}) \to 0$ für $m \to \infty$, denn dann ist $p_m n_{m-1}$ sogar summierbar über m. Also stimmt (3.2.2) in diesem Fall. Wenn $\mathbb{E}(Y_K) = \infty$, so ist auf Grund obiger Rechnung auch $\sum_{j=1}^{\infty} p_j \alpha(k_j) = \infty$, also ist (3.2.2) auch dann erfüllt. □

Wir betrachten nun kurz allgemeinere Strategien. Zunächst nehmen wir an, dass Läufe unterbrochen und später fortgesetzt werden können, d. h. in Stufe $i \in \mathbb{N}$ kann der Lauf m_i für k_i Schritte an der Stelle, an der er vorher unterbrochen wurde, wieder fortgesetzt werden und wird dann wieder unterbrochen. Hierbei haben wir also eine *gemischte* Strategie $(m_i, k_i)_{i \in \mathbb{N}}$ und haben mehr Möglichkeiten zur Optimierung. Als zweite Erweiterungsmöglichkeit können wir die Parameter m_i und k_i als *zufällig* annehmen, allerdings unter der Bedingung, dass m_i und k_i nicht von den zukünftigen Parametern $m_{i+1}, k_{i+1}, \dots$ abhängen dürfen.

Ohne Beweis (siehe [LSZ93]) versichern wir hier nur, dass eine geeignete Modifizierung des Beweises von Satz 3.2.1 zeigt, dass auch unter allen gemischten, zufälligen Strategien die konstante Strategie $K \equiv (k)_{i \in \mathbb{N}}$ mit $k \in A$ wie in Satz 3.2.1 optimal ist.

3.3 Strategien bei unbekannter Verteilung

Die Ergebnisse der Abschnitte 3.1-3.2 sind von geringem praktischen Nutzen, da die Verteilung von X meist im Dunkeln liegt. In diesem Abschnitt werden wir eine *universelle* Strategie angeben (d. h. eine, die unabhängig von der Verteilung von X definiert wird), so dass die erwartete Laufzeit nur um einen logarithmischen Term größer ist als die optimale von Satz 3.2.1, die ja von der Verteilung von X abhängt. Ferner zeigen wir, dass diese Strategie in dem Sinne optimal ist, dass die Größenordnung ihrer erwarteten Laufzeit erreicht werden kann von geeigneten Verteilungen.

Wir stellen nun diese universelle Strategie vor. Sei $K = (k_i)_{i \in \mathbb{N}}$ eine Folge von Zweierpotenzen mit der Eigenschaft, dass

$$\#\big\{i \in \{1, \dots, 2^m - 1\} \colon k_i = 2^j\big\} = 2^{m-1-j}, \qquad 0 \leq j < m. \tag{3.3.1}$$

In Worten: Für jedes $m \in \mathbb{N}$ tritt unter den $2^m - 1$ ersten Zahlen $k_1, \dots, k_{2^m-1}$ die Zahl 2^{m-1} genau einmal auf, die Zahl 2^{m-2} genau zwei Mal, die Zahl 2^{m-3} genau vier Mal und so weiter, also die Zahl 1 genau 2^{m-1} Mal. Ein Beispiel ist die Folge $K = (1, 1, 2, 1, 1, 2, 4, 1, 1, 1, 1, 2, 2, 4, 8, \dots)$. Eine solche Strategie nennen wir *ausbalanciert*. Diese Eigenschaft impliziert, dass zu jedem Zeitpunkt von der Form $k_1 + \dots + k_{2^m-1}$ die Gesamtzeit, die der Algorithmus bisher mit Durchläufen einer gegebenen Länge verbringt, nicht von dieser Länge abhängt, sondern nur von m. Wenn wir definieren $n_j = \sum_{i=1}^{j} k_i$, dann gilt insbesondere

$$n_{2^m-1} = \sum_{j=0}^{m-1} 2^j 2^{m-j-1} = m2^{m-1}, \qquad m \in \mathbb{N}. \tag{3.3.2}$$

Wie bisher bezeichnen wir für eine $\mathbb{N} \cup \{\infty\}$-wertige Zufallsvariable X mit $\alpha^* = \alpha^*(X) = \min_{k \in \mathbb{N} \cup \{\infty\}} \frac{\mathbb{E}(X \wedge k)}{\mathbb{P}(X \leq k)}$ die minimale Erwartung der Laufzeit eines jeweils nach fester Schrittzahl gestoppten und neugestarteten Algorithmus. Wir zeigen nun, dass die universelle Strategie K in (3.3.1), angewendet auf eine beliebige solche Verteilung X, eine nicht allzu hohe erwartete Laufzeit für den so gestoppten und neugestarteten Prozess ergibt.

Satz 3.3.1. *Es gibt Konstanten $c_1, c_2 > 0$, so dass für jede ausbalancierte Strategie $K = (k_i)_{i \in \mathbb{N}}$ und für jede $\mathbb{N} \cup \{\infty\}$-wertige Zufallsvariable X gilt:*

$$\mathbb{E}(Y_K) \leq c_1 \alpha^*(X)(c_2 + \log \alpha^*(X)).$$

Man kann im Beweis sehen, dass zum Beispiel $c_1 = 180$ und $c_2 = 4$ ausreichen, aber wir machen keinen Versuch, diese Werte zu optimieren.

Bevor wir Satz 3.3.1 beweisen, führen wir die nützliche Größe

$$\beta^* = \min_{k \in \mathbb{N}} \beta(k), \qquad \text{wobei } \beta(k) = \frac{k}{\mathbb{P}(X \leq k)},$$

ein. Das Minimum wird angenommen, da $\lim_{k \to \infty} \beta(k) = \infty$. Wie α^* ist auch β^* eine Funktion der Verteilung von X. Es zeigt sich, dass der Quotient β^*/α^* gleichmäßig beschränkt ist:

Lemma 3.3.2. *Für alle Verteilungen $\mathcal{L}(X)$ gilt $\alpha^* \leq \beta^* \leq 4\alpha^*$.*

Beweis. Die erste Ungleichung ist offensichtlich, da $\mathbb{E}(X \wedge k) \leq k$ für jedes $k \in \mathbb{N}$. Wir beweisen nun die zweite. Für jedes $k \in \mathbb{N}$ gilt

$$\mathbb{E}(X \wedge k) > \frac{k}{2} \qquad \text{oder} \qquad \mathbb{P}(X \leq \lfloor 2\mathbb{E}(X \wedge k) \rfloor) \geq \frac{1}{2},$$

denn falls $2\mathbb{E}(X \wedge k) \leq k$, dann folgt mit Hilfe der Markov-Ungleichung $\mathbb{P}(X > \lfloor 2\mathbb{E}(X \wedge k) \rfloor) = \mathbb{P}(X \wedge k > \lfloor 2\mathbb{E}(X \wedge k) \rfloor) = \mathbb{P}(X \wedge k > 2\mathbb{E}(X \wedge k)) \leq \frac{1}{2}$.

Sei nun $k \in A$. Im ersten Fall folgt

$$\alpha^* = \frac{\mathbb{E}(X \wedge k)}{\mathbb{P}(X \leq k)} > \frac{k}{2\mathbb{P}(X \leq k)} \geq \frac{\beta^*}{2}.$$

Im zweiten Fall folgt

$$\beta^* \leq \frac{\lfloor 2\mathbb{E}(X \wedge k) \rfloor}{\mathbb{P}(X \leq \lfloor 2\mathbb{E}(X \wedge k) \rfloor)} \leq 4\mathbb{E}(X \wedge k) \leq 4\frac{\mathbb{E}(X \wedge k)}{\mathbb{P}(X \leq k)} = 4\alpha^*.$$

Dies beendet den Beweis. □

Beweis von Satz 3.3.1. Wir wählen ein $k^* \in \mathbb{N}$ mit $\beta^* = \beta(k^*)$. Die Intuition ist die folgende. Die Wahrscheinlichkeit, dass der Algorithmus zu einem Zeitpunkt noch nicht terminiert hat, zu dem er schon $2^\ell \beta(k^*)$ Läufe der Länge $\approx k^*$ gemacht hat, ist extrem gering, wenn ℓ groß wird, so dass wir ℓ als endlich, aber genügend groß annehmen können. Auf Grund der Ausbalanciertheit sind zu diesem Zeitpunkt aber auch schon für jedes $j \leq \log_2(2^\ell \beta(k^*))$ jeweils etwa $2^\ell \beta(k^*)$ Zeiteinheiten mit Schritten der Länge 2^j verbracht worden, also insgesamt schon etwa $2^\ell \beta(k^*) \log[2^\ell \beta(k^*)] = \mathcal{O}(\beta^* \log_2(\beta^*)) + \mathcal{O}(\beta^*)$ Zeiteinheiten. Auf Grund von Lemma 3.3.2 ist dies nicht größer als $\mathcal{O}(\alpha^* \log(\alpha^*)) + \mathcal{O}(\alpha^*)$, wie behauptet.

Wir kommen zu den Details. Auf Grund von (3.3.2) ist

$$\mathbb{P}\big(Y_K > m2^{m-1}\big) = \mathbb{P}(Y_K > n_{2^m-1}) \leq \mathbb{P}(X > 2^j)^{2^{m-1-j}}, \qquad 0 \leq j < m. \tag{3.3.3}$$

Wir setzen $j_0 = \lceil \log_2 k^* \rceil$ und $n_0 = \lceil -\log_2 \mathbb{P}(X \leq k^*) \rceil$. (Man beachte, dass $2^{j_0+n_0} \approx \beta^*$.) Wir dürfen voraus setzen, dass $j_0 + n_0 \geq 1$, denn sonst ist $Y_K = 1$ mit Wahrscheinlichkeit Eins. Von (3.3.3) erhalten wir für alle $\ell \in \mathbb{N}_0$:

$$\begin{aligned}
\mathbb{P}\big(Y_K > (j_0 + n_0 + \ell + 1)2^{j_0+n_0+\ell}\big) &\leq \mathbb{P}(X > 2^{j_0})^{2^{n_0+\ell}} \\
&\leq \big(1 - \mathbb{P}(X \leq k^*)\big)^{2^\ell/\mathbb{P}(X \leq k^*)} \\
&\leq \exp\{-2^\ell\}.
\end{aligned}$$

Daraus folgt

$$\begin{aligned}
\mathbb{E}(Y_K) &\leq (j_0 + n_0 + 1)2^{j_0+n_0} + \sum_{\ell \in \mathbb{N}_0} (j_0 + n_0 + \ell + 2)2^{j_0+n_0+\ell+1} \exp\{-2^\ell\} \\
&\leq (j_0 + n_0)2^{j_0+n_0} 2\Big(1 + \sum_{\ell \in \mathbb{N}_0} (\ell + 3)\Big(\frac{2}{e^2}\Big)^\ell\Big).
\end{aligned} \tag{3.3.4}$$

Offensichtlich konvergiert die Reihe. Auf Grund der Wahl von j_0 und n_0 haben wir

$$2^{j_0+n_0} \leq 2^{\log_2 k^* - \log_2 \mathbb{P}(X \leq k^*) + 2} = 4\frac{k^*}{\mathbb{P}(X \leq k^*)} = 4\beta^*.$$

Wenn wir dies in (3.3.4) einsetzen und Lemma 3.3.2 beachten, erhalten wir die Aussage. □

Nun zeigen wir, dass die Abschätzung in Satz 3.3.1 optimal ist in dem Sinne, dass es Verteilungen von X gibt, so dass die Größenordnung des zugehörigen α^* gleich der Ordnung der oberen Schranke in Satz 3.3.1 ist.

Satz 3.3.3. *Für jede Strategie $K = (k_i)_{i \in \mathbb{N}}$ und für jedes $\alpha \in [1, \infty)$ gibt es eine Zufallsvariable X, so dass das zugehörige Y_K erfüllt: $\mathbb{E}(Y_K) \geq \frac{1}{8}\alpha \log_2 \alpha$.*

Beweis. Wir konstruieren eine endliche Folge von Verteilungen, so dass das jeweilige α^* gleich α ist und so dass für mindestens eine von ihnen der erwartete Wert des zugehörigen Y_K nicht kleiner als $\frac{1}{8}\alpha \log \alpha$ ist.

Setze $n^* = \lfloor \log_2 \alpha \rfloor$. Für $j = 0, \dots, n^*$ setze (beachte, dass $2^j/\alpha \leq 1$)

$$\mathbb{P}_j(X = k) = \begin{cases} 2^j/\alpha, & \text{falls } k = 2^j, \\ 1 - 2^j/\alpha, & \text{falls } k = \infty, \\ 0 & \text{sonst.} \end{cases}$$

In Worten: Der j-te dieser Algorithmen terminiert mit Wahrscheinlichkeit $2^j/\alpha$ nach genau 2^j Schritten und gerät ansonsten in eine Endlosschleife. Man sieht leicht, dass $\mathbb{E}_j(X \wedge 2^j) = 2^j$ und dass der zum j-ten Algorithmus gehörige α^*-Wert erfüllt: $\alpha_j^* = \alpha$.

Sei nun $K = (k_i)_{i\in\mathbb{N}}$ beliebig. Wir definieren für $t \in \mathbb{N}$ die Anzahl der bis zum Zeitpunkt t beendeten Läufe der Mindestlänge 2^j durch

$$c(j,t) = \#\{i \in \{1,\dots,m\} \colon k_i \geq 2^j\} + 1\!\!1\{t - n_m \geq 2^j\}, \qquad \text{falls } n_m \leq t < n_{m-1},$$

wobei wir daran erinnern, dass $n_i = \sum_{j=1}^i k_j$. (Man beachte, dass $c(j,t)$ nicht von den $n^* + 1$ oben definierten Verteilungen abhängt.) Also gilt für jedes $t \in \mathbb{N}$ und alle $j = 0,\dots,n^*$:

$$\mathbb{P}_j(Y_K > t) = \mathbb{P}_j(X > 2^j)^{c(j,t)} = (1 - 2^j/\alpha)^{c(j,t)}.$$

Wegen der Ungleichung $(1-x)^{\frac{1}{2x}} \geq \frac{1}{2}$ für jedes $x \in (0, \frac{1}{2}]$ gilt

$$\mathbb{P}_j(Y_K > t) \geq \frac{1}{2}, \qquad \text{falls } c(j,t) \leq \frac{\alpha}{2^{j+1}},$$

also insbesondere

$$\mathbb{E}_j(Y_K) \geq \frac{t}{2}, \qquad \text{falls } c(j,t) \leq \frac{\alpha}{2^{j+1}}.$$

Zusammenfassend haben wir

$$\max_{j=0}^{n^*} \mathbb{E}_j(Y_K) \geq \frac{t}{2}$$

für jedes $t > 0$ mit der Eigenschaft, dass es ein $j \in \{0,\dots,n^*\}$ gibt mit $c(j,t) \leq \alpha/2^{j+1}$, d. h. mit der Eigenschaft, dass für ein $j \in \{0,\dots,n^*\}$ höchstens $\alpha/2^{j+1}$ Läufe der Mindestlänge 2^j bis zum Zeitpunkt t durchlaufen werden. Wir zeigen nun, dass $t = \frac{1}{4}\alpha \log \alpha$ diese Eigenschaft hat, womit der Beweis beendet wird.

Falls nämlich $c(j,t) \geq \alpha/2^{j+1}$ für jedes $j = 0,\dots,n^*$, so gilt

$$\begin{aligned} t &\geq \sum_{j=0}^{n^*-1} \big(c(j,t) - c(j+1,t)\big)2^j + c(n^*,t)2^{n^*} = c(0,t) + \sum_{j=1}^{n^*} 2^{j-1} c(j,t) \\ &\geq \frac{\alpha}{2} + \sum_{j=1}^{n^*} 2^{j-1} \frac{\alpha}{2^{j+1}} = \alpha \frac{n^*+2}{4} \geq \frac{1}{4}\alpha \log_2 \alpha, \end{aligned}$$

wobei wir uns an die Definition von n^* erinnert haben. □

Kapitel 4

Fingerabdrücke

Zwei Strings $x, y \in A^n$ der (großen) Länge n mit Koeffizienten aus einer endlichen Menge A sollen auf Gleichheit überprüft werden. Die deterministische Komplexität dieser Aufgabe (d. h. wenn man die Strings Bit für Bit mit einander vergleicht) ist von der Ordnung n. Ein randomisierter Algorithmus, den wir in diesem Kapitel vorstellen wollen, testet nur die Bilder von x und y unter einer gewissen zufälligen Abbildung $F\colon A^n \to B$ auf Gleichheit, wobei B eine wesentlich kleinere Menge als A^n ist. Die Übereinstimmung der sogenannten *Fingerabdrücke* $F(x)$ und $F(y)$ lässt sich unter viel geringerem Aufwand prüfen. Im negativen Fall weiß man definitiv, dass $x \neq y$. Im positiven Fall bleibt eine Möglichkeit, dass $x \neq y$, obwohl $F(x) = F(y)$. Die Kunst liegt darin, geeignete Verteilungen von Fingerabdrücken F zu finden, sodass mit gleichmäßig hoher Wahrscheinlichkeit von $F(x) = F(y)$ auf $x = y$ geschlossen werden kann.

4.1 Die Technik von Freivalds

Wir wollen entscheiden, ob $AB = C$ für drei gegebene $n \times n$-Matrizen A, B und C gilt. Die Bildung des Produkts AB und der komponentenweise Vergleich ist uns zu teuer, der schnellste bekannte deterministische Algorithmus zur Multiplikation zweier $n \times n$-Matrizen hat die Laufzeit $\mathcal{O}(n^{2{,}376})$. Eine alternative Möglichkeit bietet die folgende, auf Freivalds [F77] zurück gehende Technik.

Probabilistischer Test für Gleichheit von Matrizenprodukten.
Eingabe: drei $n \times n$-Matrizen A, B und C über einem Körper $\mathbb{F}$.

Hilfsmittel: beliebige endliche Teilmenge $\mathcal{S}$ von $\mathbb{F}$.

Test auf $AB = C$:

1. Wähle einen Vektor $R \in \mathcal{S}^n$ zufällig aus.
2. Bilde die Vektoren $X = BR$, $Y = AX$ und $Z = CR$.
3. Falls $Y \neq Z$, dann verkünde '$AB \neq C$'. Falls $Y = Z$, dann vermute '$AB = C$'.

Zur Berechnung von X, Y und Z werden jeweils n^2 Schritte benötigt. Die Antwort '$AB \neq C$' ist definitiv richtig. Die Wahrscheinlichkeit, dass im Fall $Y = Z$ das ausgegebene Urteil '$AB = C$' falsch ist, ist gering:

Lemma 4.1.1. *Es seien A, B und C beliebige Matrizen mit $AB \neq C$. Dann gilt für einen zufälligen, auf $\mathcal{S}^n$ gleichförmig verteilten Vektor R die Abschätzung $\mathbb{P}(ABR = CR) \leq |\mathcal{S}|^{-1}$.*

Beweis. Die Matrix $D = AB - C = (d_{ij})_{i,j=1,\dots,n}$ ist nach Voraussetzung ungleich der Nullmatrix. Also gibt es Indices i und j mit $d_{ij} \neq 0$. Falls $DR = 0$, so ist insbesondere $\sum_k d_{ik} R_k = 0$, also

$$R_j = -\frac{\sum_{k \neq j} d_{ik} R_k}{d_{ij}}. \tag{4.1.1}$$

Die Koeffizienten $R_1, \dots, R_n$ sind unabhängige, auf $\mathcal{S}$ gleichförmig verteilte Zufallsgrößen. Bei gewählten R_k mit $k \neq j$ (d. h., bedingt auf die Zufallsgrößen R_k mit $k \neq j$) hat das Ereignis in (4.1.1) also höchstens die Wahrscheinlichkeit $|\mathcal{S}|^{-1}$. Also ist auch die unbedingte Wahrscheinlichkeit des Ereignisses höchstens $|\mathcal{S}|^{-1}$. □

Bemerkung 4.1.2. Durch unabhängiges k-maliges Ziehen des Vektors R in $\mathcal{S}^n$ kann man die Wahrscheinlichkeit einer falschen Entscheidung auf $|\mathcal{S}|^{-k}$ drücken.

4.2 Testen der Gleichheit von Strings

Zwei Personen A und B möchten die Gleichheit zweier langer Strings, also 0-1-Folgen, auf ihre Gleichheit hin überprüfen. A und B sind mit einander durch einen zuverlässigen, aber teuren Kanal verbunden. Die simple Möglichkeit, den String $a = (a_1, \dots, a_n) \in \{0,1\}^n$ an B zu schicken und dann bitweise mit dem String $b = (b_1, \dots, b_n)$ zu vergleichen, ist zu teuer. Das folgende randomisierte Verfahren reduziert die Übertragungsraten erheblich und fällt mit hoher Wahrscheinlichkeit das richtige Urteil. Wir identifizieren die Strings a und b mit den Zahlen $\sum_{i=1}^n a_i 2^{i-1} \in \mathbb{N}_0$ bzw. $\sum_{i=1}^n b_i 2^{i-1}$. Für jede Primzahl p bezeichnen wir mit $F_p \colon \mathbb{N}_0 \to \{0, 1, \dots, p-1\}$ die Abbildung $F_p(x) = x \mod p$.

Randomisierter Gleichheitstest.

Eingabe: $a, b \in \{0, \dots, 2^n\}$.

Parameter: $k \in \mathbb{N}$.

Algorithmus:

1. Person A wählt zufällig eine Primzahl X aus $\{2, \dots, k\}$ und berechnet $F_X(a)$.
2. Person A schickt X und $F_X(a)$ an B.
3. Person B berechnet $F_X(b)$ und vergleicht es mit $F_X(a)$.
4. Falls $F_X(a) \neq F_X(b)$, so verkündet B, dass $a \neq b$. Falls $F_X(a) = F_X(b)$, so vermutet B, dass $a = b$.

Der ‘Fingerabdruck’ ist also der Rest beim Teilen durch eine zufällige Primzahl. In gewissem Sinn ist dieses Verfahren dual zu dem Verfahren von Freivalds in Abschnitt 4.1: Dort ist der Körper $\mathbb{F}$ fixiert, und man randomisiert die Wahl der Stelle, an der man die Matrix auswertet. Hier wertet man an einer festen Stelle aus, aber randomisiert die Wahl des Fingerabdrucks.

Das Urteil ist korrekt, wenn $F_X(a) \neq F_X(b)$. Die Fehlerwahrscheinlichkeit im Fall $F_X(a) = F_X(b)$ ist gering:

Satz 4.2.1. *Seien $a, b \in \{0, 1, 2, \dots, 2^n\}$ mit $a \neq b$, und sei X eine zufällige Primzahl in $\{2, \dots, k\}$, dann gilt*

$$\mathbb{P}\big(F_X(a) = F_X(b)\big) \leq \frac{n \log k}{k \log n}(\log 2 + o(1)) \qquad \text{für } n \wedge k \to \infty. \tag{4.2.1}$$

Beweis. Falls für eine Primzahl p gilt: $F_p(a) = F_p(b)$, so ist p ein Teiler von $|a - b|$, d. h. p ist ein Primfaktor von $j = |a - b|$. Sei $g(j)$ die Anzahl der verschiedenen Primfaktoren von $j \in \mathbb{N}$, und sei $\pi(k)$ die Anzahl der Primzahlen in $\{2, \dots, k\}$, dann folgt aus der obigen Überlegung:

$$\mathbb{P}\big(F_X(a) = F_X(b)\big) \leq \frac{1}{\pi(k)} \max_{j=1}^{2^n} g(j). \tag{4.2.2}$$

Der berühmte Primzahlsatz sagt, dass $\pi(k) \sim \frac{k}{\log k}$ für $k \to \infty$. Eine obere Schranke für $\max_{j=1}^{2^n} g(j)$ erhält man aus der Beobachtung, dass die Implikation $g(j) \geq m \Longrightarrow j \geq m!$ gilt, denn wenn j mindestens m verschiedene Primfaktoren besitzt, ist j mindestens das Produkt der kleinsten m Primzahlen, welches wir großzügig gegen $m!$ nach unten abschätzen. Aus dieser Implikation folgt die Implikation $\max_{j=1}^{2^n} g(j) = m \Longrightarrow m! \leq 2^n$. Mit Hilfe der Stirlingschen Formel ($m! \sim (\frac{m}{e})^m \sqrt{2\pi m} = (\frac{m}{e})^m (1 + o(1)) \equiv \varphi(m)$ für $m \to \infty$) erhält man die asymptotische Abschätzung

$$\max_{j=1}^{2^n} g(j) \leq \varphi^{-1}\big(2^n(1 + o(1))\big) = \log 2 \frac{n}{\log n}(1 + o(1)), \qquad n \to \infty.$$

Wenn man dies und den Primzahlsatz in (4.2.2) einsetzt, erhält man direkt die Behauptung. □

Wählt man $k = cn$ mit einem $c > 1$, dann ist die Fehlerwahrscheinlichkeit durch $\frac{1}{c} + o(1)$ nach oben beschränkt, mit $k = n^{1+\alpha}$ mit einem $\alpha > 0$ sogar durch $\mathcal{O}(n^{-\alpha})$. In beiden Fällen ist die Anzahl der zu übertragenden Bits lediglich von der Ordnung $\log n$. Das Ziehen der zufälligen Zahl X verursacht keine nennenswerten Kosten, wenn k polynomial in n ist: Man kann einfach eine beliebige zufällige Zahl in $\{1, \dots, k\}$ wählen und sie auf ihre Primzahleigenschaft testen; siehe auch Abschnitt 4.4.

4.3 Das Auffinden von Mustern

Ein klassisches und oft auftretendes Problem (man denke nur an Suchmaschinen im Internet) ist das Finden eines Ausdrucks in einem langen Text. Gegeben seien ein langer und ein kurzer String $y = (y_1, \dots, y_n)$ und $z = (z_1, \dots, z_m)$ mit $m \ll n$. Die Aufgabe ist die Bestimmung aller Positionen, an denen der Ausdruck z im Text y vorkommt. Eine mögliche, aber teure Lösung ist, alle m-Strings $y_{(i)} = (y_i, \dots, y_{i+m-1})$ mit $1 \leq i \leq n - m + 1$ Bit für Bit mit z zu vergleichen; dies hat die Laufzeit $\mathcal{O}(mn)$. Der folgende randomisierte Algorithmus (siehe [KR87]) basiert auf dieser plumpen Methode, aber statt die $y_{(i)}$ bitweise mit z zu vergleichen, testet er, ob die Fingerabdrücke $F_X(y_{(i)})$ und $F_X(z)$ mit einander übereinstimmen, wobei $y_{(i)}$ und z, ähnlich wie in Abschnitt 4.2, mit einer natürlichen Zahl identifiziert werden (siehe unten), X eine geeignet zufällig gewählte Primzahl ist und $F_p(x) = x \mod p$.

Randomisierte Suche in einem Text.
Eingabe: zwei Strings $y = (y_1, \dots, y_n)$ und $z = (z_1, \dots, z_m)$ mit $m \leq n$.

Ausgabe: (möglichst alle) Stellen i mit $y_{(i)} = z$.

Hilfsmittel: Parameter $k \in \mathbb{N}$.

Algorithmus:

1. Wähle eine Primzahl X zufällig aus $\{2, \dots, k\}$ und berechne $F_X(z)$.
2. Für $i = 1, \dots, n - m + 1$, berechne $F_X(y_{(i)})$ und vergleiche mit $F_X(z)$. Falls $F_X(y_{(i)}) = F_X(z)$, so melde 'z befindet sich an Stelle i'.

Damit der Algorithmus sich gegenüber dem kompletten bitweisen Vergleich lohnt, sollte k nur so groß sein, dass $\log k \ll m$. Ein wesentlicher Aspekt ist, dass die Fingerabdrücke $F_X(y_{(1)}), \dots, F_X(y_{(n-m+1)})$ sehr einfach rekursiv berechnet werden können. Wenn wir $y_{(i)}$ mit $\sum_{j=0}^{m-1} y_{i+m-1-j} 2^j$ identifizieren, dann gilt

$$y_{(i+1)} = 2y_{(i)} - 2^m y_i + y_{i+m},$$

also

$$F_p(y_{(i+1)}) = 2F_p(y_{(i)}) - 2^m y_i + y_{i+m} \mod p.$$

Insbesondere ist die Anzahl der arithmetischen Operationen konstant bei diesen Schritten.

Wenn k geeignet gewählt ist, dann ist auch die Wahrscheinlichkeit, dass der Algorithmus fälschlicherweise eine Übereinstimmung meldet, gering:

Satz 4.3.1. *Mit einem $\alpha > 0$ wählen wir $k = mn^{1+\alpha}$. Dann ist die Wahrscheinlichkeit, dass der Algorithmus eine nicht vorhandene Übereinstimmung meldet, höchstens von der Ordnung $n^{-\alpha}$.*

Beweis. Der Algorithmus meldet eine nicht vorhandene Übereinstimmung, wenn ein i existiert mit $F_X(y_{(i)}) = F_X(z)$, obwohl $y_{(i)} \neq z$, wenn also X ein Teiler von $|y_{(i)} - z|$ ist für ein i. Dies ist genau dann der Fall, wenn X ein Teiler von $\prod_{i\colon y_{(i)} \neq z} |y_{(i)} - z|$ ist, welches nicht größer als 2^{mn} ist. Auf die selbe Weise wie im Beweis von Satz 4.2.1 erhält man für die Fehlerwahrscheinlichkeit asymptotisch die obere Schranke

$$\frac{\log k}{k} \log 2 \frac{mn}{\log(mn)} = n^{-\alpha} \log 2 \frac{\log(mn^{1+\alpha})}{\log(mn)} = \mathcal{O}(n^{-\alpha}).$$

□

Durch wiederholtes Ausführen des Algorithmus mit unabhängigen Primzahlen kann die Fehlerwahrscheinlichkeit weiter gedrückt werden. Man kann den Algorithmus auch leicht in einen Las-Vegas-Algorithmus verwandeln (also eine in jedem Fall korrekte Antwort erzwingen), wenn man für jedes i mit $F_X(y_{(i)}) = F_X(z)$ Bit für Bit die Übereinstimmung von $y_{(i)}$ mit z vergleicht. Die erwartete Laufzeit dieses Algorithmus ist $\mathcal{O}(n + \ell m)$, wobei ℓ die Anzahl der tatsächlichen Übereinstimmungen bezeichnet.

4.4 Primzahltests

Bei den in den Abschnitten 4.2 und 4.3 vorgestellten Algorithmen werden Primzahlen benötigt, und zwar potentiell sehr große. Auch in der Kryptografie werden große Primzahlen benutzt, um

z. B. einen Verschlüsselungscode sicher zu machen. Verschlüsselungen werden auch weithin im Internet benutzt, etwa beim Homebanking. Wenn man z. B. zwei sehr große Primzahlen p und q besitzt, dann basieren die effektivsten Verschlüsselungsmethoden auf der Tatsache, dass man zunächst als Einziger im Besitz der Primzerlegung der Zahl $n = pq$ ist, und wenn p und q wirklich groß sind, dann wird das auch noch eine Weile so bleiben, denn die systematische Findung der Zerlegung $n = pq$ ist ein höchst rechenaufwendiges Problem. Die gigantische Komplexität dieser Aufgabe ist es, die die Verschlüsselung vor dem Knacken schützt.

Bemerkung 4.4.1 (Das RSA-Verfahren). Das meist benutzte Verschlüsselungsverfahren, das sogenannte *RSA-Verfahren* (benannt nach den Initialen der Autoren von [RSA78]), funktioniert folgendermaßen. Mit zwei geheimen, sehr großen Primzahlen p und q veröffentlicht man das Produkt $n = pq$. Ebenfalls veröffentlicht man den sogenannten *öffentlichen Schlüssel* e. Der Klartext m ist eine Zahl in $\{0, 1, \dots, n-1\}$, die man zu $c = m^e \mod n$ verschlüsselt. Der Code c wird gesendet und braucht nicht geheim gehalten zu werden. Der Empfänger ist im Besitz des geheimen persönlichen Schlüssels d, der die Eigenschaft $de \mod [\varphi(n)] = 1$ hat, wobei $\varphi(n) = (p-1)(q-1)$ ist. Nach dem Satz von Euler gilt $m^{k\varphi(n)+1} \mod n = m$ für jedes $m \in \{0, 1, \dots, n-1\}$ und jedes $k \in \mathbb{N}$. Zur Entschlüsselung des Codes c berechnet der Empfänger $c^d \mod n = m^{de} \mod n = m$ und erhält den Klartext.

Um den Code zu brechen, müsste ein Dritter entweder den privaten Schlüssel d aus e berechnen oder auf irgendeine Weise $\varphi(n)$ errechnen. Es lässt sich zeigen, dass Beides mindestens so schwierig ist, wie die Primfaktorzerlegung $n = pq$ zu berechnen. ◇

Da bei jedem kryptografischen Protokoll neue Primzahlen benötigt werden, sind Primzahltests äußerst wichtig, denn Primzahlen beschafft man sich gewöhnlich, indem man zufällig eine Zahl zieht und dann sie einen Primzahltest bestehen lässt: Falls sie das nicht tut, versucht man es mit einer anderen. Leider ist es sehr zeitraubend, systematisch etwa mit dem *Sieb des Erathostenes* eine zu untersuchende Zahl n durch alle Zahlen $\leq \sqrt{n}$ zu teilen und zu prüfen, ob der Rest immer nicht Null ist. Der Aufwand dieser sicheren Methode ist exponentiell in der Anzahl der Stellen von n.

In diesem Abschnitt diskutieren wir drei probabilistische Primzahltests, die alle nach dem folgenden Prinzip funktionieren. Es sei eine große ungerade Zahl n gegeben, die wir auf ihre Primeigenschaft testen wollen. Wir wählen einen zufälligen *Zeugen* a aus, mit dem wir n auf eine gewisse Eigenschaft testen, meist auf das Bestehen einer gewissen Gleichung zwischen a und n, die mindestens immer dann erfüllt ist, wenn n prim ist. Im negativen Fall steht also sofort fest, dass n zusammengesetzt ist. Im positiven Fall kann man noch nichts schließen, denn auch zusammengesetzte Zahlen n erfüllen sie für manche Zeugen a. Die Kunst besteht darin, die Gleichung so zu wählen, dass für jede zusammengesetzte Zahl n die Wahrscheinlichkeit, dass der zufällig gewählte Zeuge a die Gleichung nicht erfüllt, wegbeschränkt von Null ist, dass also die Nicht-Primeigenschaft von n mit positiver Wahrscheinlichkeit erkannt wird. In dem Fall, dass die Gleichung erfüllt ist, kann man dann also vermuten, dass n prim ist, und unterliegt nur mit einer festen, von 1 wegbeschränkten Wahrscheinlichkeit einem Irrtum. Mehrfache unabhängige Wiederholung drückt die Irrtumswahrscheinlichkeit geometrisch. In unseren Beispielen wird der Zeuge a gleichverteilt aus der Menge $\{1, \dots, n-1\}$ gewählt, also sollte die zu testende Gleichung bei zusammengesetztem n für einen gewissen positiven Anteil von Zeugen $a \in \{1, \dots, n-1\}$ falsch werden, damit der Algorithmus die Nicht-Primeigenschaft von n mit hoher Wahrscheinlichkeit erkennt.

In Abschnitt 4.4.1 stellen wir einen Prototyp vor, der dieser Anforderung nur knapp nicht entspricht. In den Abschnitten 4.4.2 und 4.4.3 geben wir dann probabilistische Tests, die diesen Mangel überwunden haben und heute zu den meist benutzten Primzahltests gehören.

4.4.1 Der Fermat-Test

Ausgangspunkt der Primzahltests ist der folgende berühmte Satz:

Satz 4.4.2 (Kleiner Fermatscher Satz). *Sei p eine Primzahl. Dann gilt für alle $a \in \mathbb{N}$ mit* $\operatorname{ggT}\{a,p\} = 1$ *die Gleichung* $a^{p-1} = 1 \mod p$.

Diese Aussage liefert den folgenden Primzahltest:

Fermatscher Primzahltest.

Eingabe: $n \in \mathbb{N}$ ungerade.

Parameter: $k \in \mathbb{N}$.

Test:

1. Wiederhole die folgenden Schritte (a) – (b) unabhängig k Male:
   (a) Wähle zufällig $a \in \{1, \dots, n-1\}$ und prüfe, ob $\operatorname{ggT}\{a,n\} = 1$. Falls nicht, so verkünde 'n ist zusammengesetzt' und terminiere.
   (b) Prüfe, ob $a^{n-1} = 1 \mod n$. Falls dies nicht zutrifft, so verkünde 'n ist zusammengesetzt' und terminiere.
2. Vermute 'n ist prim'.

Wenn der Algorithmus verkündet, dass n zusammengesetzt sei, so stimmt dies garantiert. Doch im gegenteiligen Fall hat der Algorithmus die folgende empfindliche, unbehebbare Schwäche. Betrachten wir die Menge aller Restklassen, für die der Fermatsche Test kein Ergebnis liefert:

$$P_n = \{a \in \mathbb{Z}_n^* \colon a^{n-1} = 1 \mod n\}.$$

Man kann leicht zeigen, dass P_n eine Untergruppe der Gruppe der primen Restklassen $\mathbb{Z}_n^*$ ist und daher (nach dem Satz von Lagrange) $|P_n|$ ein Teiler von $|\mathbb{Z}_n^*|$ ist. Falls also n nicht prim ist und $P_n \neq \mathbb{Z}_n^*$, so gilt $|\mathbb{Z}_n^* \setminus P_n| \geq n/2$, und der Fermat-Test erkennt mit Wahrscheinlichkeit von mindestens $\frac{1}{2}$, dass n nicht prim ist. Wenn wir in diesem Fall den Test also k Mal mit unabhängigen, auf $\{1, \dots, n-1\}$ gleichförmig verteilten $a_1, \dots, a_k$ durchführen und das Ergebnis jedes Mal $\operatorname{ggT}\{a,n\} = 1$ und $a^{n-1} = 1 \mod n$ ist, und wir schließen daraus auf Primeigenschaft von n, so ist die Wahrscheinlichkeit für einen Irrtum höchstens 2^{-k}. So weit, so gut für Kandidaten n mit $P_n \neq \mathbb{Z}_n^*$.

Falls aber n zusammengesetzt ist und $P_n = \mathbb{Z}_n^*$, so kann der Fermat-Test n nicht von einer Primzahl unterscheiden. Solche Zahlen n gibt es tatsächlich, und zwar nennt man sie (per definitionem) die *Carmichael-Zahlen*. Es gibt unendlich viele davon, wie man seit 1992 weiß, wenn sie auch relativ dünn gesät sind: Unter den ersten 10^{15} Zahlen gibt es nur etwa 10^5 Carmichael-Zahlen. Trotzdem impliziert die Existenz der Carmichael-Zahlen die prinzipielle Untauglichkeit des Fermat-Tests. Der Fermat-Test ist also kein Las-Vegas-Algorithmus. Man beachte allerdings, dass der Fermat-Test sehr viel schneller ist als der systematische Erathostenes-Test, denn er ist nur polynomial in der Anzahl der Stellen von n.

4.4.2 Der Solovay-Strassen-Test

In [SS77] präsentierten Solovay und Strassen eine Weiterentwicklung des Fermat-Tests, der auf quadratischen Resten beruht. Wir benötigen ein paar Begriffe. Im Folgenden sei n eine ungerade Zahl. Eine Zahl $a \in \mathbb{Z}_n^*$ heißt *quadratischer Rest*, wenn es eine Restklasse $x \in \mathbb{Z}_n^*$ gibt, so dass $x^2 = a$. Anderenfalls heißt a ein *quadratischer Nichtrest*. Man kann leicht zeigen, dass es höchstens $\frac{1}{2}|\mathbb{Z}_n^*|$ quadratische Reste gibt, also mindestens $\frac{1}{2}|\mathbb{Z}_n^*|$ quadratische Nichtreste. Für eine Primzahl p und eine Zahl $a \in \mathbb{Z}_p^*$ definieren wir das *Legendre-Symbol*

$$\left(\frac{a}{p}\right) = \begin{cases} +1, & \text{falls } a \text{ ein quadratischer Rest ist,} \\ -1 & \text{sonst.} \end{cases}$$

Die Definition des Legendre-Symbols kann man erweitern zur Definition des *Jacobi-Symbols*, indem man für eine ungerade Zahl n mit Primfaktorenzerlegung $n = p_1^{a_1} \cdots p_k^{a_k}$ setzt:

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{a_i}.$$

Die Berechung des Jacobisymbols $\left(\frac{a}{n}\right)$ erfordert glücklicherweise *nicht* die Kenntnis der Primfaktorenzerlegung von n. Es gibt einen Algorithmus zur Berechnung mit einem Aufwand, der höchstens quadratisch in der Anzahl der Stellen von n ist. Dies kann man mit Hilfe der Rechenregeln

$$\begin{aligned} \left(\frac{ab}{n}\right) &= \left(\frac{a}{n}\right)\left(\frac{b}{n}\right), \qquad a, b \in \mathbb{Z}_n^*, \\ \left(\frac{2}{n}\right) &= (-1)^{\frac{n^2-1}{8}}, \\ \left(\frac{p}{q}\right) &= (-1)^{\frac{p-1}{2}\frac{q-1}{2}}\left(\frac{q}{p}\right), \qquad p, q \text{ prim} \end{aligned}$$

einsehen. Der Ausgangspunkt des Solovay-Strassen-Tests ist das folgende Ergebnis:

Satz 4.4.3. *Für jede Primzahl $n > 2$ und alle Zahlen $a \in \mathbb{Z}_n^*$ gilt:*

$$a^{\frac{n-1}{2}}\left(\frac{a}{n}\right) = 1 \mod n. \tag{4.4.1}$$

Falls n keine Primzahl ist, so ist die Anzahl der $a \in \mathbb{Z}_n^*$, sodass (4.4.1) *nicht erfüllt ist, mindestens $\frac{n}{2}$.*

Beweis. Die erste Aussage ist eine direkte Folgerung aus dem Kleinen Fermatschen Satz. In [MR95, Lemma 14.8] wird die zweite Aussage bewiesen. (Um dies zu zeigen, reicht es, zu zeigen, dass für zusammengesetzes n mindestens ein a existiert, für das (4.4.1) nicht erfüllt ist, denn die komplementäre Menge ist eine Untergruppe, also ist ihre Kardinalität ein Teiler von n.) □

Insbesondere existiert auch für jede Carmichael-Zahl n immer eine Zahl $a \in \mathbb{Z}_n^*$, sodass entweder ggT $\{a, n\} \neq 1$ gilt (dann ist n offensichtlich nicht prim) oder (4.4.1) nicht erfüllt ist. Ein Test, der auf (4.4.1) basiert, besitzt also nicht die strukturelle Schwäche des Fermat-Tests.

Auf Satz 4.4.3 basiert der folgende Primzahltest:

Solovay-Strassen-Primzahltest.
Eingabe: eine ungerade Zahl $n \in \mathbb{N}$.

Test:

1. Wiederhole die folgenden Schritte (a) – (c) unabhängig k Male:
   (a) Wähle zufällig $a \in \{1, \dots, n-1\}$ und prüfe, ob ggT $\{a, n\} = 1$. Falls nein, so verkünde 'n ist zusammengesetzt' und terminiere.
   (b) Berechne $x = a^{\frac{n-1}{2}} (\frac{a}{n})$.
   (c) Falls $x \neq 1 \mod n$, so verkünde 'n ist zusammengesetzt' und terminiere.
2. Vermute, dass n prim ist.

Falls der Algorithmus verkündet, dass n zusammengesetzt ist, stimmt dies garantiert. Falls er vermutet, dass n prim ist, hat er auf Grund von Satz 4.4.3 mit einer Wahrscheinlichkeit $\geq 1-2^{-k}$ Recht. Damit ist der Solovay-Strassen-Primzahltest ein echter Las-Vegas-Algorithmus. Wie oben erwähnt, ist der Aufwand zur Berechnung des Jacobi-Symbols bezüglich n von der Ordnung des Quadrats der Anzahl der Stellen von n, also viel geringer als der Aufwand beim Sieb des Erathostenes.

4.4.3 Der Rabin-Miller-Test

Der Rabin-Miller-Test (siehe [R80]) beruht auf der folgenden Verfeinerung des Fermat-Tests: Sei n die zu testende Zahl, natürlich eine ungerade. Sei

$$n - 1 = 2^t \cdot q, \qquad t \in \mathbb{N} \text{ maximal}, \, q \in \mathbb{N},$$

die Zerlegung von $n-1$ in eine Zweierpotenz 2^t und einen ungeraden Anteil q. Wir ziehen einen *Zeugen* $a \in \{2, \dots, n-1\}$. Falls ggT $\{a, n\} \neq 1$ ist, so ist n offensichtlich zusammengesetzt, wir könnnen also annehmen, dass ggT $\{a, n\} = 1$. Wir betrachten

$$b = a^q \mod n \qquad \text{und} \qquad s = \min\{r < t \colon (b^{2^r})^2 = 1 \mod n\}.$$

Das Minimum ist wohldefiniert, denn nach dem Kleinen Fermatschen Satz liegt mindestens $t-1$ in der Menge, über die das Minimum gebildet wird: Es gilt ja $(b^{2^{t-1}})^2 = a^{2^t q} = a^{n-1} = 1 \mod n$. Nun benötigen wir das folgende Ergebnis:

Satz 4.4.4. *Falls n prim ist, so gilt entweder $a^q = 1 \mod n$ oder $b^{2^s} = -1 \mod n$ für alle $a \in \{2, \dots, n-1\}$. Falls n zusammengesetzt ist, ist die Anzahl der $a \in \{2, \dots, n-1\}$ mit $a^q \neq 1 \mod n$ und $b^{2^s} \neq -1 \mod n$ mindestens $\frac{n}{2}$.*

Beweisskizze. Die Reihe der $q+1$ Zahlen $a^q, a^{2q}, a^{4q}, a^{8q}, \dots, a^{2^t q} = a^{n-1}$ (immer $\mod n$ betrachtet) endet also nach dem Kleinen Fermatschen Satz mit einer 1. Falls schon die erste dieser Zahlen, also b, gleich 1 ist, so sind alle diese Zahlen gleich 1 ($\mod n$). Ansonsten gibt es ein s (nämlich das oben definierte), so dass die $(s+1)$-te Zahl gleich 1 ist und die s-te Zahl eine Einheitswurzel ungleich 1. Wenn n prim ist, dann ist $\mathbb{Z}_n$ ein Körper, und in diesem Fall kommt als einzige Einheitswurzel ungleich 1 nur die -1 in Frage.

In dem Fall, dass n zusammengesetzt ist, muss für den Beweis der Aussage nur gezeigt werden, dass es Zeugen $a \in \{2, \ldots, n-1\}$ gibt mit $b^{2^s} \neq -1 \mod n$. Dies wird in [R80] durchgeführt. □

Wählt man also $a \in \{2, \ldots, n-1\}$ *zufällig* aus, so ist bei zusammengesetztem n die Wahrscheinlichkeit, dass das zugehörige b und s nicht n als zusammengesetzt erkennt, nicht größer als $\frac{1}{2}$. Mehrmaliges unabhängiges Ziehen von a drückt diese Fehlerwahrscheinlichkeit deutlich.

Rabin-Miller-Primzahltest.
Eingabe: Eine ungerade Zahl $n \in \mathbb{N}$.

Test:

1. Finde durch fortgesetztes Halbieren ein maximales $t \in \mathbb{N}$ und ein $q \in \mathbb{N}$ mit $n - 1 = 2^t \cdot q$.
2. Wiederhole die folgenden Schritte (a)-(d) unabhängig k Male:
   (a) Wähle zufällig $a \in \{2, \ldots, n-1\}$. Falls ggT $\{a, n\} \neq 1$, so melde 'n ist zusammengesetzt' und terminiere.
   (b) Bilde $b = a^q \mod n$. Falls $b = 1 \mod n$, vermute 'n ist prim' und terminiere.
   (c) Finde durch fortgesetztes Quadrieren das kleinste $s \in \{1, \ldots, t\}$ mit $(b^{2^s})^2 = 1 \mod n$.
   (d) Falls $b^{2^s} \neq -1 \mod n$, so verkünde 'n ist zusammengesetzt' und terminiere.
3. Vermute 'n ist prim'.

Auch dieser Test ist ein Las-Vegas-Test: Falls er zu dem Schluss kommt, dass n zusammengesetzt ist, hat er Recht, und im gegenteiligen Fall irrt er mit einer Wahrscheinlichkeit von nicht mehr als $(\frac{1}{2})^k$.

Kapitel 5

Klassische Markovketten-Monte-Carlo-Methoden

In diesem Kapitel behandeln wir das Problem, auf effiziente Weise aus einer gegebenen Menge I eine Stichprobe mit gegebener Verteilung π auf I zu ziehen. Hierbei wird I endlich sein, aber gigantisch groß, so dass die in Kapitel 1 erwähnten naiven Methoden nicht effizient sind. Wir werden in diesem Kapitel Verfahren vorstellen, die darauf basieren, dass eine geeignet gewählte Markovkette gegen ihre invariante Verteilung konvergiert. Nach 'genügend langer' Laufzeit sollte diese Verteilung also 'genügend gut' approximiert sein, und wir können abbrechen. Dieses Prinzip wird seit den 50er Jahren verwendet und untersucht, und es bringt praktischerseits gute Ergebnisse, obwohl die theoretische Absicherung immer mit großem mathematischen Aufwand verbunden ist und nur sehr pessimistische Schranken hervor bringt.

Die beiden Hauptvertreter dieser Methode sind der *Metropolis-Algorithmus* und der *Gibbs-Sampler*, die wir beide diskutieren werden. Außerdem bringen wir ein (nicht optimales) Resultat zum *Simulated Annealing*, einem verwandten Algorithmus.

5.1 Motivation

Hier sind ein paar Beispiele:

Beispiel 5.1.1 (Kartenmischen). Wir wollen einen Kartenstapel perfekt mischen. Das heißt, auf der Menge I aller Permutationen der Zahlen $1, 2, \dots, 32$ (oder gar 52) wollen wir die Gleichverteilung simulieren. $\diamond$

Beispiel 5.1.2 (Handelsreisender). Ein Handelsreisender möchte N gegebene Städte je genau einmal bereisen und sucht eine kürzeste Route (welche, ist ihm egal). Wenn wir annehmen, dass man einer Tabelle die jeweiligen Abstände $f(i,j)$ zwischen der i-ten und der j-ten Stadt entnehmen kann, so ist also für jede Reihenfolge $\sigma\colon \{1,\dots,N\} \to \{1,\dots,N\}$ die Länge der durch σ gegebenen Route, $L(\sigma) = \sum_{k=1}^{N} f(\sigma(k-1), \sigma(k))$, zu minimieren, wobei $\sigma(0) = \sigma(N)$. Natürlich nehmen wir an, dass σ eine Permutation von $\{1,\dots,N\}$ ist. Also möchten wir die Gleichverteilung auf der Menge aller Permutationen σ mit minimalem $L(\sigma)$ simulieren. $\diamond$

Beispiel 5.1.3 (statistische Physik). In der statistischen Physik benutzt man als diskrete

Modelle für physikalische Vorgänge oft Wahrscheinlichkeitsmaße der Form

$$\pi(i) = \frac{1}{Z_{n,\beta}} e^{-\beta H_n(i)}, \qquad i \in I = S^{\Lambda_n}, \tag{5.1.1}$$

wobei $\Lambda_n = \{-n, \dots, n\}^d$ eine große Box im d-dimensionalen Gitter $\mathbb{Z}^d$ ist, $i = (i_x)_{x \in \Lambda_n} \in I = S^{\Lambda_n}$ eine Konfiguration, wobei S eine endliche Menge ist (der *Spinraum*), $\beta > 0$ ein Stärkenparameter und $Z_{n,\beta}$ die Normierungskonstante (die *Zustandssumme*), so dass π ein Wahrscheinlichkeitsmaß auf I ist. Jedem Gitterpunkt $x \in \Lambda_n$ wird also ein 'Spin' $i_x \in S$ zugeordnet. Das Maß π bevorzugt Konfigurationen, die einen geringen Wert der *Hamilton-Abbildung* $H_n : I \to \mathbb{R}$ (der *Energie*) aufweisen, und dieser Effekt wird um so stärker, je größer β ist. Im Extremfall $\beta = \infty$ ist π die gleichförmige Verteilung auf der Menge aller Minimumspunkte von H_n. Im anderen Extremfall, $\beta = 0$, ist π die Gleichverteilung auf I. Man interpretiert oft β als den Kehrwert der Temperatur. Je größer β also ist, desto stärker wird das System 'abgekühlt', bis es beim absoluten Nullpunkt in den wenigen Zuständen minimaler Energie 'eingefroren' ist. Je kleiner β ist, also je heißer das System wird, desto größer wird die 'Unordnung', bis beim völlig ungeordneten System alle Konfigurationen gleich wahrscheinlich sind.

Das Ziehen einer Stichprobe mit der gegebenen Verteilung π wird auch benötigt, um gewisse charakteristische Kenngrößen empirisch festzustellen, um Aussagen über ihren ungefähren erwarteten Wert zu machen. Im Prinzip können, je nach Anwendungen, alle diese Modelle auf jedem beliebigen Grafen an Stelle von Teilmengen des $\mathbb{Z}^d$ betrachtet werden. $\diamond$

Es folgen ein paar Spezialfälle von Beispiel 5.1.3.

Beispiel 5.1.4 (Ising-Modell). Ein sehr bekanntes und oft benutztes Modell für die Magnetisierung eines Eisens in einem Magnetfeld, das sogenannte *Ising-Modell*, erhält man z. B. durch die Wahl $S = \{-1, 1\}$ und

$$H_n(i) = -\sum_{x,y \in \Lambda_n} v_{x,y} i_x i_y - h \sum_{x \in \Lambda_n} i_x,$$

wobei $v_{x,y} \in \mathbb{R}$ die Interaktion zwischen den Ladungen in den Gitterpunkten x und y ist und $h \in \mathbb{R}$ die Wirkung eines externen Feldes beschreibt..

Im Folgenden beschränken wir uns auf den Fall, wo $v_{x,y} = 1$, wenn x und y Nachbarn sind, und $v_{x,y} = 0$ sonst, und wir schalten das externe Feld aus, d. h. wir setzen $h = 0$. Hier ist

$$H_n = \sum_{x,y \in \Lambda_n :\, x \sim y} i_x i_y$$

(wir schreiben $x \sim y$, wenn x und y Nachbarn sind) die Differenz der Anzahl der Nachbarpunktpaare mit verschiedener Ladung und der mit gleicher Ladung, und diese Differnenz soll also möglichst gering sein. Der Hamiltonian ist also klein, wenn in dem Eisen die Ladungen benachbarter Teilchen mit einander übereinstimmen. Ein Grund für das hohe Interesse am Ising-Modell ist die Tatsache, dass es einen *Phasenübergang* aufweist: Es gibt einen kritischen Wert β_c, so dass im Grenzübergang $n \to \infty$ für $\beta < \beta_c$ die Abhängigkeiten der Spins so klein sind, dass ein Gesetz der Großen Zahlen gilt, d. h. die Relation der $+1$-Spins zu den -1-Spins gleich ist. Für $\beta > \beta_c$ allerdings setzt sich die Interaktion durch, und nur diejenigen Konfigurationen werden auftreten, in denen gleiche Spins weit überwiegen. Physikalisch interpretiert man dieses Phänomen als *spontane Magnetisierung*.

Die Zustandssumme $Z_{n,\beta}$ enthält physikalisch relevante Informationen, wie die *mittlere Energie*,

$$\mathbb{E}_\pi(H_n) = \frac{1}{Z_{n,\beta}} \sum_{i\in I} H_n(i) e^{-\beta H_n(i)} = -\frac{1}{Z_{n,\beta}} \frac{\partial}{\partial\beta} Z_{n,\beta} = -\frac{\partial}{\partial\beta} \log Z_{n,\beta},$$

oder die *mittlere Magnetisierung*,

$$\sum_{i\in I} \frac{1}{|\Lambda_n|} \Big(\sum_{x\in\Lambda_n} i_x\Big) \pi(i) = \frac{1}{|\Lambda_n|} \frac{1}{Z_{n,\beta}} \sum_{i\in I} \Big(\sum_{x\in\Lambda_n} i_x\Big) e^{-\beta H_n(i)} = \frac{1}{|\Lambda_n|} \frac{1}{\beta} \frac{\partial}{\partial b} \log Z_{n,\beta}.$$

Allgemeine exakte Formeln für $Z_{n,\beta}$ gibt es nur in Dimensionen $d \in \{1,2\}$, ansonsten ist $Z_{n,\beta}$ unzugänglich. Also ist es von Interesse, approximative Simulationen für die Verteilung π zu haben. ◇

Beispiel 5.1.5 (Hard-Core-Modell). Ein Teilchenmodell mit Abstoßung erhält man durch die Wahl $S = \{0,1\}$, $\beta = \infty$ und $H_n(i) = \sum_{x,y\in\Lambda_n : |x-y|=1} i_x i_y$, denn dann ist π die Gleichverteilung auf der Menge aller 0-1-Konfigurationen, in denen je zwei benachbarten Gitterpunkten nicht gleichzeitig der Wert 1 zugeordnet wird. Einen Gitterpunkt x mit $i_x = 1$ interpretiert man dann als 'besetzt', einen mit $i_x = 0$ als 'vakant'. Man interessiert sich für die Anzahl der Konfigurationen und für die erwarteten Anzahl der Moleküle in einer Konfiguration. ◇

Beispiel 5.1.6 (Self-avoiding walk). Mit einer leichten Modifikation erhält man Pfadmodelle, wenn man I ersetzt durch die Menge aller Nächstnachbarschaftspfade, $\{(i_0,\dots,i_n) \in (\mathbb{Z}^d)^{n+1} : i_0 = 0, \forall x : |i_x - i_{x+1}| = 1\}$. Mit der Wahl $H_n(i) = \sum_{0\le x<y\le n} \mathbb{1}\{i_x = i_y\}$, der Anzahl der Selbstüberschneidungen, erhält man ein Modell für einen selbstüberschneidungsfreien Pfad für $\beta = \infty$ (dies ist der so genannte *self-avoiding walk*, die Gleichverteilung auf der Menge aller Pfade ohne Doppelpunkte) bzw. für einen Pfad mit unterdrückten Selbstüberschneidungen für $\beta < \infty$, den so genannten *self-repellent walk*. Das Hauptinteresse gilt der Anzahl der selbstüberschneidungsfreien Pfade (also der Zustandssumme) und dem erwarteten Abstand der beiden Enden der Kette als eine Maßzahl für die räumliche Ausbreitung der Kette. ◇

Man beachte, dass in allen vorgestellten Beispielen die Verteilung π nur durch *relative* Gewichtung definiert wird, indem man jedem Punkt aus I eine Maßzahl zuordnet (bei Gleichverteilung ist dies einfach 1) und dann durch die Gesamtsumme aller Gewichtungen teilt.

Eine grundsätzliche Herangehensweise, die wir in diesem Kapitel vorstellen, basiert auf der Tatsache, dass gewisse Markovketten bei immer längerer Laufzeit gegen ihre invariante Verteilung konvergieren. Man konstruiert also zu der Verteilung π eine geeignete Markovkette, deren invariante Verteilung π ist, und dann lässt man diese Kette so lange laufen, bis man vermutet, das sie 'nahe genug' an der Verteilung π ist. Wir stellen zunächst die wichtigsten Tatsachen über Markovketten bereit.

5.2 Markovketten

Ohne Beweise präsentieren wir diejenigen Aussagen über Markovketten, die wir hier benötigen werden. Mehr Material über Markovketten findet man etwa in [No97] oder [H02].

Eine *Markovkette* auf I mit Übergangsmatrix $P = (p_{i,j})_{i,j\in I}$ und Startverteilung ν auf I ist eine endliche oder unendliche Folge von I-wertigen Zufallsgrößen $X_0, X_1, X_2, \dots$ mit der

Eigenschaft, dass für alle $n \in \mathbb{N}_0$ und alle $i_0, i_1, \dots, i_n \in I$ gilt:

$$\mathbb{P}(X_0 = i_0, X_1 = i_1, \dots, X_n = i_n) = \nu(i_0) \prod_{k=1}^{n} p_{i_{k-1}, i_k}.$$

Hierbei ist P eine *stochastische Matrix*, d. h. eine Matrix mit nichtnegativen Einträgen, so dass die Summe über jede Zeile Eins ergibt: $\sum_{j \in I} p_{i,j} = 1$ für jedes $i \in I$. Der Eintrag $p_{i,j}$ heißt die *Übergangswahrscheinlichkeit* von i nach j. Das Maß $\mathbb{P}$ wird auch mit $\mathbb{P}_\nu$ bezeichnet, um die Startverteilung ν zu betonen. Falls $\nu = \delta_i$ für ein $i \in I$, also wenn die Kette in i startet, dann schreibt man auch $\mathbb{P}_i$ statt $\mathbb{P}_{\delta_i}$.

Man nennt die Markovkette oder die Übergangsmatrix P *irreduzibel*, falls $\mathbb{P}_i(\exists n \in \mathbb{N}\colon X_n = j) > 0$ für alle $i, j \in I$, d. h. wenn jeder Punkt von jedem aus mit positiver Wahrscheinlichkeit erreichbar ist. Mit $p_{i,j}^{(n)}$ bezeichnen wir die Koeffizienten der n-ten Potenz der Matrix P. Eine irreduzible Matrix P heißt *aperiodisch*, falls die *Periode* eines jeden $i \in I$, also die Zahl $d_i = \mathrm{ggT}\,\{n \in \mathbb{N}\colon p_{i,i}^{(n)} > 0\}$, gleich Eins ist. Eine stochastische Matrix P ist genau dann aperiodisch und irreduzibel, wenn es ein n_0 gibt, so dass P^n für jedes $n \geq n_0$ ausschließlich positive Einträge hat. Wenn ein Zustand $i^* \in I$ existiert, so dass $p_{i^*,i^*} > 0$ und so dass die Kette mit positiver Wahrscheinlichkeit von jedem Startpunkt aus i^* erreichen kann, so ist P irreduzibel und aperiodisch.

Eine Verteilung π auf I heißt *invariant unter P*, wenn die Gleichung $\pi = \pi P$ (im Sinne des Matrixprodukts) gilt. Dies ist genau dann der Fall, wenn die mit der Startverteilung π gestartete Kette zu jedem Zeitpunkt die Verteilung π besitzt. Falls eine invariante Verteilung π existiert, so gilt für jede Startverteilung ν:

$$\lim_{n \to \infty} \mathbb{P}_\nu(X_n = j) = \pi(j), \qquad j \in I. \tag{5.2.1}$$

Eine Verteilung π ist insbesondere invariant, wenn sie *reversibel* ist, d. h. wenn die Gleichung

$$\pi(i) p_{i,j} = \pi(j) p_{j,i}, \qquad i, j \in I, \tag{5.2.2}$$

gilt.

5.3 Der Metropolis-Algorithmus

Einer der meist benutzten Ansätze zur approximativen Ziehung einer wie π verteilten Stichprobe ist die Konstruktion einer Markovkette auf I, die π als eine invariante, besser noch: als reversible, Verteilung besitzt. Auf Grund von (5.2.1) approximiert die Kette bei beliebiger Startverteilung dann die Verteilung π nach genügend langer Laufzeit.

Eine der meist benutzten geeigneten Standard-Markovketten für dieses Problem ist der sogenannte *Metropolis-Algorithmus*, siehe [Ha70] und [MRRTT53]. Hierbei wird zunächst eine dem konkreten Problem angepasste 'Referenz-Übergangsmatrix' $Q = (q_{i,j})_{i,j \in I}$ gewählt, die aperiodisch und irreduzibel ist und 'viele Nullen' enthält, also nur relativ wenige Übergänge zulässt. Dann definiert man die

Metropolis-Matrix $\boldsymbol{P}$:

$$p_{i,j} = \begin{cases} q_{i,j} \min\Big\{1, \frac{\pi(j) q_{j,i}}{\pi(i) q_{i,j}}\Big\}, & \text{falls } i \neq j \text{ und } q_{i,j} > 0,\\ 0, & \text{falls } i \neq j \text{ und } q_{i,j} = 0,\\ 1 - \sum_{k \in I \setminus \{i\}} p_{i,k}, & \text{falls } i = j. \end{cases} \tag{5.3.1}$$

Dann ist offensichtlich π die reversible Verteilung von P. Wenn Q irreduzibel ist und π nicht die Gleichverteilung, dann ist P auch irreduzibel und aperiodisch. Man beachte, dass nur die Quotienten $\pi(j)/\pi(i)$ bekannt sein müssen, um P zu berechnen. Eine geschickte Wahl von Q benutzt oft eine gewisse Nachbarschaftsstruktur auf I: Sie wählt einen Zustand j aus, wenn er in geeignetem Sinn benachbart ist zum aktuellen Zustand i, d. h. wenn j sich nur sehr wenig unterscheidet von i und daher die Berechnung von $\pi(j)/\pi(i)$ nur wenige Rechenoperationen erfordert.

Beispiel 5.3.1 (Simulation bedingter Verteilungen). Seien $I_1, I_2, I_3, \dots$ unabhängige $\mathbb{N}_0$-wertige Zufallsvariablen mit Verteilung $(p_x)_{x \in \mathbb{N}_0}$. Wir wollen die bedingte Verteilung des Vektors $(I_1, \dots, I_n)$ gegeben $\{I_1 + \dots + I_n = m\}$ für gegebenes $m \in \mathbb{N}_0$ simulieren. Wir wählen also den Zustandsraum

$$I = \Big\{(i_1, \dots, i_n) \in \mathbb{N}_0^n \colon \sum_{x=1}^n i_x = m\Big\},$$

und die zu simulierende Verteilung ist

$$\pi(i) = \frac{1}{c_m} \prod_{x=1}^n p_{i_x}, \qquad \text{wobei } c_m = \sum_{j \in I} \prod_{x=1}^n p_{j_x}.$$

Wir fassen $i, j \in I$ als benachbart auf, wenn sie sich höchstens in zwei Komponenten unterscheiden, wenn also $x \neq y \in \{1, \dots, n\}$ existieren mit $i_z = j_z$ für alle $z \in \{1, \dots, n\} \setminus \{x, y\}$. Also hat jeder Punkt $i \in I$ genau $\sum_{1 \leq x < y \leq n} (i_x + i_y) = m(n-2)$ Nachbarn. Eine geeignete Wahl der Referenzmatrix Q ist die folgende. Zunächst wählt man zufällig zwei Punkte $x, y \in \{1, \dots, n\}$. Falls $x \neq y$, und falls $i \in I$ der aktuelle Zustand ist, so wähle ein zufälliges $k \in \{0, \dots, i_x + i_y\}$. Dann ersetzt man i_x durch $j_x = k$ und i_y durch $j_y = i_x + i_y - k$. Nach Konstruktion ist dann $j = (j_z)_{z=1,\dots,n}$ in I.

Diese Referenzkette ist irreduzibel, falls z. B. der Träger $\{y \in \mathbb{N}_0 \colon p_y > 0\}$ ein Intervall ist. Der Algorithmus akzeptiert ein vorgeschlagenes j mit Wahrscheinlichkeit 1, falls $\pi(j) \geq \pi(i)$, bzw. mit Wahrscheinlichkeit

$$\frac{\pi(j)}{\pi(i)} = \frac{p_{j_x} p_{j_y}}{p_{i_x} p_{i_y}} \in [0, 1]$$

sonst. ◇

Beispiel 5.3.2 (Ising-Modell). Im Ising-Modell (siehe Beispiel 5.1.4) wählen wir die folgende natürliche Nachbarschaftsstruktur: Die Menge der Nachbarn einer Konfiguration $i \in I = \{-1, 1\}^{\Lambda_n}$ ist deie Menge aller Konfigurationen, die sich in genau einem Spin von i unterscheiden, also

$$\mathcal{N}_i = \{j \in I \setminus \{i\} \colon \text{es gibt ein } x \in \Lambda_n \text{ so dass } i_{-x} = j_{-x}\},$$

wobei

$$i_{-x} = \big\{j = (j_y)_{y \in \Lambda_n} \colon j_y = i_y \text{ für alle } y \neq x\big\}. \tag{5.3.2}$$

Es gilt offensichtlich $|\mathcal{N}_i| = |\Lambda_n|$, und die Äquivalenz $j \in \mathcal{N}_i \iff i \in \mathcal{N}_j$ ist auch leicht zu sehen. Die Matrix Q ist also gegeben durch $q_{i,j} = 1/|\Lambda_n|$, falls $j \in \mathcal{N}_i$, und $q_{i,j} = 0$ sonst, und Q ist symmetrisch. Dann hat die Metropolis-Matrix P also die Form

$$p_{i,j} = \frac{1}{|\Lambda_n|} \min\bigl\{1, e^{-\beta(H_n(j)-H_n(i))}\bigr\} = \frac{1}{|\Lambda_n|} e^{-\beta(H_n(j)-H_n(i))^+}.$$

P bevorzugt also den Übergang zu Konfigurationen mit geringerer Energie, und diese Bevorzugung wird desto stärker, je größer β ist. Man sieht leicht, dass im Fall $i_x = -1$ und $j_x = +1$ gilt:

$$H_n(j) - H_n(i) = 4[k_+(i,x) - k_-(i,x)], \qquad i \in I, j \in \mathcal{N}_i \text{ und } i_{-x} = j_{-x},$$

wobei $k_+(i,x)$ und $k_-(i,x)$ die Anzahl der $+1$-Spins bzw. der -1-Spins in der Konfiguration in den Nachbarpunkten von x ist: Da $j_z = i_z$ für alle $z \in \Lambda_n \setminus \{x\}$, gilt

$$\begin{aligned} H_n(j) - H_n(i) &= - \sum_{y\sim z\colon y=x \text{ oder } z=x} \bigl[j_z j_y - i_z i_y\bigr] = -2 \sum_{z\in\Lambda_n\colon z\sim x} (j_x - i_x) i_z \\ &= 4 \sum_{z\in\Lambda_n\colon z\sim x} i_z = 4[k_+(i,x) - k_-(i,x)]. \end{aligned}$$

Insbesondere hängen die Übergangswahrscheinlichkeiten bei gewähltem Gitterpunkt nur von den Spins der Konfiguration in den Nachbarn ab; ihre Berechnung erfordert also keinen großen Aufwand. $\diamond$

5.4 Gibbs-Sampler

Für die Simulation von Verteilungen π auf Zustandsräumen der Form $I = S^V$ (mit einem endlichen 'Spinraum' S und einer endlichen 'Gitter' V) wird oft ein Algorithmus benutzt, der *Gibbs-Sampler* genannt wird und nahe Verwandtschaft zum Metropolis-Algorithmus aufweist. In Worten kann man den Gibbs-Sampler wie folgt zusammenfassen. Es sei ein Zustand $i = (i_x)_{x\in V} \in I$ gegeben.

Der Gibbs-Sampler.

(i) Wähle eine zufällige Komponente $x \in V$ aus (oft nach der Gleichverteilung).

(ii) Ersetze die x-te Komponente von i durch einen zufälligen Wert nach der bedingten Verteilung von π gegeben, dass alle anderen Komponenten gleich denen von i sind, d. h. im gleichverteilten Fall nach der Verteilung

$$i_{-x} \ni j \mapsto \frac{\pi(j)}{\sum_{j'\in i_{-x}} \pi(j')}, \qquad \text{wobei } i_{-x} = \bigl\{j = (j_y)_{y\in V} \colon j_y = i_y \text{ für alle } y \neq x\bigr\}.$$

(iii) Übernehme alle anderen Komponenten von i.

Schritt (i) kann mit Hilfe einer Referenz-Übergangsmatrix Q modelliert werden, wie beim Metropolis-Algorithmus. Wenn wir die selbe Wahl wie in Beispiel 5.3.2 treffen, dann erhalten wir die Gibbs-Sampler-Übergangsmatrix P, die gegeben ist als

$$p_{i,j} = \frac{1}{|V|} \frac{\pi(j)}{\sum_{j' \in i_{-x}} \pi(j')}, \qquad \text{falls } i_{-x} = j_{-x}.$$

Es ist klar, dass die so definierte Markovkette ebenfalls π als reversible Verteilung besitzt. Die Irreduzibilität muss von Fall zu Fall geklärt werden.

Beispiel 5.4.1 (Ising-Modell). Wir betrachten das Ising-Modell; siehe Beispiele 5.1.4 und 5.3.2. Wir wenden den Gibbs-Sampler an, und zwar mit der Gleichverteilung in Schritt (i). In diesem Fall hat die Übergangsmatrix die Form

$$p_{i,j} = \frac{1}{|\Lambda_n|} \big(1 + e^{\beta(H_n(j) - H_n(i))}\big)^{-1}, \qquad \text{falls } j \in \mathcal{N}_i. \tag{5.4.1}$$

Auch der Gibbs-Sampler hat also die Tendenz, zu Konfigurationen geringerer Energie überzugehen. Der Term $(H_n(j) - H_n(i))$ tauchte auch in der entsprechenden Metropolis-Matrix auf, siehe die Bemerkungen dazu in Beispiel 5.3.2. $\diamond$

5.5 Eine Konvergenzabschätzung

Um den Markovketten-Algorithmus lohnend anzuwenden, braucht man Abschäzungen für die Geschwindigkeit seiner Konvergenz. Im Allgemeinen sind solche Abschätzungen nicht leicht zu erhalten, und diejenigen, die man mathematisch beweisen kann, sind meist sehr pessimistisch. Die optimale Größenordnung (in Termen der Größe des Zustandsraumes) ist meist relativ leicht zu erhalten, aber die involvierten Konstanten hängen oft von komplizierten Kenngrößen der 'Topographie' des Zustandsraumes ab und sind daher weit entfernt von der optimalen.

In diesem Abschnitt bringen wir die exakte Formulierung eines Konvergenzresultats für eine Variante des Gibbs-Samplers in Anwendung auf ein gewisses Modell der statistischen Physik.

Beispiel 5.5.1 (Der systematische Gibbs-Sampler). Statt in Schritt (i) des Algorithmus den Spin, den man neu auswürfeln möchte, zufällig nach der Gleichverteilung zu wählen, kann man ihn auch systematisch in einer festgelegten Reihenfolge wählen. Falls also $V = \{x_1, x_2, \dots, x_K\}$ eine Aufzählung des Grafen V ist (mit $K = |V|$), dann kann man in Schritt (i) die Ecke x_i z. B. zu den Zeitpunkten $i, K+i, 2K+i, \dots$ auswählen und dann Schritt (ii) auf x_i anwenden. Diese Modifikation ergibt eine zeitlich *inhomogene* Markovkette; man nennt diesen Algorithmus den *systematic sweep Gibbs Sampler.* Man sieht leicht, dass auch diese Markovkette aperiodisch und irreduzibel ist und die selbe invariante Verteilung besitzt wie der gewöhnliche Gibbs-Sampler. $\diamond$

Beispiel 5.5.2 (Zufällige Färbung eines Grafen). Eine Verallgemeinerung des Hard-Core-Modells in Beispiel 5.1.5 erhält man, wenn man den Spinraum $\{0,1\}$ durch eine beliebige endliche Menge $S = \{1, \dots, q\}$ ersetzt. Die Konfigurationen in S^V können dann aufgefasst werden als zufällige Färbungen des Grafen V mit q Farben. Die Bedingung, die wir stellen, ist, dass keine zwei benachbarten Ecken des Grafen die selbe Farbe besitzen. Die Gleichverteilung auf der Menge dieser Konfigurationen kann man formal in der Form (5.1.1) fassen mit $\beta = \infty$ und einem Hamiltonian $H_n \geq 0$, der genau dann Null ist, wenn die Konfiguration die obige Eigenschaft hat. $\diamond$

Nun bringen wir also ein theoretisches Resultat über die Anwendung des systematischen Gibbs-Samplers auf das Grafenfärbungsmodell. Genauer gesagt, wir geben eine Schranke für die Anzahl der benötigten Iterationen der Markovkette, um mit ihrer Verteilung innerhalb einer gegebenen (kleinen) Umgebung der invarianten Verteilung π anzukommen. Wir benötigen noch eine Metrik auf der Menge aller Verteilungen auf I, hier wählen wir die *Totalvariationsnorm*, also

$$\|\mu_1 - \mu_2\|_{\mathrm{TV}} = \sup_{A \subset I} |\mu_1(A) - \mu_2(A)| = \frac{1}{2} \sum_{i \in I} |\mu_1(i) - \mu_2(i)| \in [0,1] \qquad (5.5.1)$$

für Wahrscheinlichkeitsmaße μ_1, μ_2 auf I.[1]

Satz 5.5.3. *Es sei $G = (V, E)$ ein Graf mit $k = |V|$ Knoten, so dass jeder Knoten höchstens d Nachbarn hat. Wir betrachten das Grafenfärbungsmodell aus Beispiel 5.5.2 mit $q > 2d^2$. Sei $\varepsilon > 0$ gegeben. Dann ist die Anzahl der Iterationen, die der systematische Gibbs-Sampler aus Beispiel 5.5.1 benötigt (startend von einer beliebigen festen Konfiguration), um innerhalb der ε-Umgebung von π in Totalvariation zu sein, höchstens*

$$k\Big(\frac{\log k + \log(\varepsilon^{-1}) - \log d}{\log(\frac{q}{2d^2})}\Big).$$

Für einen Beweis siehe [H02, Theorem 8.1]. Wir beschränken uns auf ein paar Bemerkungen.

Bemerkung 5.5.4. (a) Die Anzahl der benötigten Iterationen ist also von der Ordnung $k \log k$ (in der Größe des Zustandsraums) bzw. $k \log(\varepsilon^{-1})$, was nicht furchtbar groß erscheint.

(b) Die Bedingung $q \geq d^2$ kann zu $q > 2d$ abgeschwächt werden, siehe [Je95]. Zum Beispiel hat man dann schnelle Konvergenz im Fall des Grafen $V = \mathbb{Z}^2$ (dann ist $d = 4$) schon für mindestens zehn Farben, statt nach Satz 5.5.3 erst für $q > 32$.

(c) Auf dem ersten Blick scheint es widersinnig, dass der Algorithmus für *große* q besser funktioniert, denn dann sollte der Algorithmus doch langsamer sein, weil man viele Farben mit einander mischen muss. Die korrekte Intuition ist aber, dass bei großer Farbenzahl q die Farbe eines gegebenen Knotens weniger abhängig wird von der Farbe seiner Nachbarn: Z. B. könnte man bei sehr großem q die jeweilige Farbe einfach gleichmäßig aus $\{1, \dots, q\}$ ziehen und hat nur geringes Risiko, eine der Farben der Nachbarn zu treffen.

$\diamond$

5.6 Simulated Annealing

Simulated Annealing ist ein populärer Algorithmus, der approximativ das Minimum einer gegebenen Funktion auf einer endlichen Menge findet. Eines der Standardbeispiele ist das Problem des Handelsreisenden (Beispiel 5.1.2), aber auch Packungsprobleme (gegeben viele Kisten unterschiedlicher Abmessungen, wie packe ich sie möglichst Platz sparend in eine gegebene Halle?). Wir können die Notationen von Beispiel 5.1.3 teilweise übernehmen: Es sei $H \colon I \to \mathbb{R}$ eine reellwertige Funktion auf einer endlichen Menge I, und wir suchen $H_* = \min_{i \in I} H(i)$ und einen

[1] Der Beweis der Formel in (5.5.1) ist eine Übungsaufgabe.

Minimierer, also ein Element aus $M = \{i \in I \colon H(i) = H_*\}$. Wir möchten gerne die Gleichverteilung auf der Menge M simulieren. Betrachten wir das Maß π_β, das durch $\pi_\beta(i) = \frac{1}{Z_\beta} e^{-\beta H(i)}$ gegeben ist, wie in (5.1.1). Für große β approximiert π_β die gewünschte Verteilung:

Lemma 5.6.1. *Für $\beta \to \infty$ konvergiert π_β gegen die Gleichverteilung auf M, die wir mit π_∞ bezeichnen. Für ein geeignetes $\beta_0 > 0$ ist die Abbildung $\beta \mapsto \pi_\beta(i)$ auf $[\beta_0, \infty)$ sogar monoton fallend für $i \in I \setminus M$ und monoton steigend für $i \in M$.*

Beweis. Wir können Zähler und Nenner mit $e^{\beta H_*}$ multiplizieren und sehen, dass

$$\pi_\beta(i) = \frac{1}{\widetilde{Z}_\beta} e^{-\beta \widetilde{H}(i)}, \qquad i \in I,$$

wobei $\widetilde{H} = H - H_*$, und $\widetilde{Z}_\beta$ ist die geeignete Normierungskonstante. Es ist klar, dass $\widetilde{Z}_\beta = |M| + \sum_{i \in I \setminus M} e^{-\beta \widetilde{H}(i)}$. Wenn nun $\beta \to \infty$ gelassen wird, so gilt offensichtlich $\widetilde{Z}_\beta \to |M|$ und daher $\pi_\beta(i) \to 0$ für jedes $i \in I \setminus M$, denn $\widetilde{H}(i) > 0$.

Die Zusatzaussage beweisen wir durch Differenziation nach β:

$$\frac{\mathrm{d}}{\mathrm{d}\beta} \pi_\beta(i) = \pi_\beta(i) \Big[\frac{1}{\widetilde{Z}_\beta} \sum_{j \in I} \widetilde{H}(j) e^{-\beta \widetilde{H}(j)} - \widetilde{H}(i) \Big].$$

Da der erste Summand in der Klammer für $\beta \to \infty$ verschwindet, ist die gesamte Klammer negativ für alle $i \in I \setminus M$ und alle genügend großen β. Außer dem ist er immer nichtnegativ für $i \in M$, denn dann ist $\widetilde{H}(i) = 0$. □

Daraus ergibt sich die folgende Idee: Wenn wir für jedes $\beta > 0$ eine Markovkette haben (etwa die Metropolis-Kette), die die Verteilung π_β approximiert, und wenn wir sukzessive β immer größer machen, dann sollte doch diese inhomogene Kette gegen π_∞ konvergieren. Dies ist die Idee des Simulated Annealing, des 'simulierten Härtens'. Der Begriff kommt aus der Metallurgie, wo durch Abkühlung Metall gehärtet wird. Die Abkühlung entspricht der Erniedrigung der Temperatur, also dem Erhöhen von β, das immer die Interpretation der inversen Temperatur besitzt (siehe auch Beispiel 5.1.3). Allerdings darf man nicht zu schnell abkühlen, denn das Metall könnte in einer unerwünschten Form erstarren, bzw. die Markovkette könnte sich in einem lokalen, aber nicht globalen, Minimum fangen lassen.

Wir wählen also eine 'Kühlungsstrategie' $(\beta_n)_{n \in \mathbb{N}}$, d. h. die Markovkette $(X_n)_{n \in \mathbb{N}}$ auf I besitzt zu jedem Zeitpunkt $n \in \mathbb{N}$ die Metropolis-Übergangsmatrix $P^{(\beta_n)}$, gegeben durch

$$p_{i,j}^{(\beta_n)} = \begin{cases} q_{i,j} e^{-\beta_n [H(j) - H(i)]^+}, & \text{falls } i \neq j, \\ 1 - \sum_{k \in I \setminus \{i\}} p_{i,k}^{(\beta_n)}, & \text{falls } i = j. \end{cases}$$

Hierbei ist $Q = (q_{i,j})_{i,j \in I}$ eine stochastische irreduzible aperiodische symmetrische Referenzmatrix.[2] Die Markovkette ist also zeitlich inhomogen, denn zum Zeitpunkt $n \in \mathbb{N}$ wird der Übergang mit Hilfe des Parameters β_n 'ausgewürfelt'. Man beachte, dass Übergänge zu energetisch weniger günstigen Zuständen geringe Wahrscheinlichkeit haben: Falls Q den Übergang von i zu j vorgeschlagen hat und $H(j) > H(i)$ ist, so wird der von Q vorgeschlagene Übergang zu j mit der

[2] Im Falle des Problems des Handelsreisenden wird man zum Beispiel Q so wählen, dass $q_{i,j} > 0$ nur eintreten kann, wenn die Permutationen i und j sich nur durch eine Transposition unterscheiden.

kleineren Wahrscheinlichkeit $e^{-\beta_n(H(j)-H(i))}$ ausgeführt, als die Wahrscheinlichkeit 1 im Falle $H(j) \leq H(i)$. Dieser Effekt ist desto stärker, je größer β_n ist. Das bedeutet, dass, je länger der Algorithmus läuft, es für die Kette immer schwerer wird, 'Pässe' zu überwinden, um in neuen 'Tälern' nach dem absoluten Minimum zu suchen. Dann sollte aber auch die Kette schon mit hoher Wahrscheinlichkeit in der Nähe des absoluten Minimums sein, damit dieser Algorithmus Erfolg hat. Hier ist ein theoretisches positives Ergebnis bei geeigneter Kühlungsstrategie:

Satz 5.6.2 ([GG84]). *Es gibt ein $\varepsilon > 0$, so dass für jede Startverteilung ν auf I und für jede Kühlungsstrategie $(\beta_n)_{n\in\mathbb{N}}$ mit $\lim_{n\to\infty} \beta_n = \infty$ und $\beta_n \leq \varepsilon \log n$ für jedes $n \in \mathbb{N}$ gilt:*

$$\lim_{n\to\infty} \mathbb{P}_\nu(X_n = i) = \pi_\infty(i), \qquad i \in I.$$

Der Beweis zeigt, dass $\varepsilon < (pK)^{-1}$ ausreichend ist, wobei

$$K = \max_{i,j\in I}[H(j) - H(i)]^+ \qquad \text{und} \qquad p \in \mathbb{N} \text{ minimal mit } q^{(p)}_{i,j}. \tag{5.6.1}$$

Beweis. Wir führen den *Dobruschinschen Kontraktionskoeffizienten* ein: Für eine stochastische Matrix $R = (R(i,j))_{i,j\in I}$ sei

$$c(R) = \sup_{i,j\in I} \|R(i,\cdot) - R(j,\cdot)\|_{\mathrm{TV}},$$

wobei $\|\cdot\|_{\mathrm{TV}}$ den Totalvariationsabstand bezeichnet; siehe (5.5.1). Der Koeffizient $c(\cdot)$ hat folgende Eigenschaften:

Lemma 5.6.3. *Für alle stochastischen Matrizen R, R_1, R_2 und alle Wahrscheinlichkeitsmaße μ_1, μ_2 auf I gelten:*

(a) $1 - c(R) = \min_{i,j\in I} \sum_{k\in I} R(i,k) \wedge R(j,k) \geq \min_{i,j\in I} R(i,j)$.

(b) $c(R_1R_2) \leq c(R_1)c(R_2)$.

(c) $\|\mu_1 R - \mu_2 R\|_{\mathrm{TV}} \leq c(R)\|\mu_1 - \mu_2\|_{\mathrm{TV}}$.

Beweis. (a) ist eine einfache Übungsaufgabe. (b) folgt leicht durch Anwendung von (c) auf $R = R_2$ und $\mu_1(\cdot) = R_1(i,\cdot)$ und $\mu_2(\cdot) = R_1(j,\cdot)$ nebst Übergang zu $\max_{i,j\in I}$.

Wir beweisen (c): Sei $A \subset I$, dann werden wir mit der Menge $B = \{j \in I : \mu_1(j) \geq \mu_2(j)\} \subset I$ zeigen, dass gilt: $\mu_1 R(A) - \mu_2 R(A) \leq c(R)(\mu_1(B) - \mu_2(B))$. Vertauschung der Rollen von μ_1 und μ_2 und Übergang zum Maximum über $A \subset I$ bzw. über $B \subset I$ lässt dann die Aussage folgen. Mit einem $i^* \in I$ so dass $R(i^*, A) = \min_{i\in I} R(i, A)$ gilt

$$\begin{aligned}
\mu_1 R(A) - \mu_2 R(A) &= \sum_{j\in I} (\mu_1(j) - \mu_2(j))(R(j,A) - R(i^*,A)) \\
&\leq \sum_{j\in B} (\mu_1(j) - \mu_2(j)) \max_{k\in I}(R(k,A) - R(i^*,A)) \\
&\leq \sum_{j\in B} (\mu_1(j) - \mu_2(j)) \max_{k\in I} \|R(k,\cdot) - R(i^*,\cdot)\|_{\mathrm{TV}} \\
&\leq c(R)(\mu_1(B) - \mu_2(B)). \quad \square
\end{aligned}$$

Wir definieren

$$R_n = P^{(\beta_1)} P^{(\beta_2)} \cdots P^{(\beta_n)}, \qquad n \in \mathbb{N},$$

also ist $\nu_n = \nu R_n$ die Verteilung der Markovkette zum Zeitpunkt n. Es genügt zu zeigen, dass

$$\lim_{n\to\infty} \|\nu_n - \pi_{\beta_n}\|_{\rm TV} = 0, \tag{5.6.2}$$

denn wegen $\beta_n \to \infty$, also $\pi_{\beta_n} \to \pi_\infty$ nach Lemma 5.6.1, folgt dann, dass $\lim_{n\to\infty} \|\nu_n - \pi_\infty\|_{\rm TV} = 0$, und dies impliziert die Behauptung. Aus notationellen Gründen schreiben wir ab jetzt $\beta(n)$ statt β_n.

Wir definieren

$$\delta_n = \sum_{k=0}^{\infty} \|\pi_{\beta(n+k+1)} - \pi_{\beta(n+k)}\|_{\rm TV}\,, \qquad n \in \mathbb{N}.$$

Man beachte, dass $\lim_{n\to\infty} \delta_n = 0$, denn für genügend großes n sind die Abbildungen $\beta \mapsto \pi_\beta(i)$ auf $[\beta_n, \infty)$ monoton fallend für $i \in I \setminus M$ und monoton steigend für $i \in M$ (siehe Lemma 5.6.1), und es gilt

$$\begin{aligned}
\delta_n &= \frac{1}{2} \sum_{k=0}^{\infty} \Big[\sum_{i\in M} (\pi_{\beta(n+k+1)}(i) - \pi_{\beta(n+k)}(i)) - \sum_{i\in I\setminus M} (\pi_{\beta(n+k+1)}(i) - \pi_{\beta(n+k)}(i)) \Big] \\
&= \|\pi_{\beta(n)} - \pi_\infty\|_{\rm TV}\,,
\end{aligned}$$

und dies verschwindet für $n \to \infty$.

Nun zeigen wir, dass für alle $n, m \in \mathbb{N}$ gilt:

$$\|\pi_{\beta(n+m)} - \nu_{n+m}\|_{\rm TV} \le c\big(P^{(\beta(n+1))} P^{(\beta(n+2))} \cdots P^{(\beta(n+m))}\big) + \delta_n. \tag{5.6.3}$$

Dies geht mit Hilfe von Lemma 5.6.3(b) und (c) folgendermaßen (wir benutzen auch die Beziehungen $\pi_\beta P^{(\beta)} = \pi_\beta$ sowie $\nu_{n+m} = \nu_{n+m-1} P^{(\beta(n+m))}$):

$$\begin{aligned}
\|\pi_{\beta(n+m)} - \nu_{n+m}\|_{\rm TV} &\le \|\pi_{\beta(n+m-1)} P^{(\beta(n+m))} - \nu_{n+m-1} P^{(\beta(n+m))}\|_{\rm TV} \\
&\quad + \|(\pi_{\beta(n+m)} - \pi_{\beta(n+m-1)}) P^{(\beta(n+m))}\|_{\rm TV} \\
&\le \|(\pi_{\beta(n+m-1)} - \nu_{n+m-1}) P^{(\beta(n+m))}\|_{\rm TV} \\
&\quad + \|\pi_{\beta(n+m)} - \pi_{\beta(n+m-1)}\|_{\rm TV} \\
&\le \cdots \le \|(\pi_{\beta(n)} - \nu_n) P^{(\beta(n+1))} \cdots P^{(\beta(n+m))}\|_{\rm TV} \\
&\qquad\qquad + \sum_{k=0}^{m-1} \|\pi_{\beta(n+k+1)} - \pi_{\beta(n+k)}\|_{\rm TV}\,,
\end{aligned}$$

und dies ist offensichtlich nicht größer als die rechte Seite von (5.6.3).

Nun zeigen wir, dass für jedes $n \in \mathbb{N}$ gilt:

$$\lim_{m\to\infty} c\big(P^{(\beta(n+1))} P^{(\beta(n+2))} \cdots P^{(\beta(n+m))}\big) = 0. \tag{5.6.4}$$

Da die Referenzmatrix Q aperiodisch und irreduzibel ist, gibt es ein minimales $p \in \mathbb{N}$, so dass $\eta = \min_{i,j\in I} q_{i,j}^{(p)} > 0$, wobei $(q_{i,j}^{(p)})_{i,j\in I}$ die p-te Potenz von Q ist; siehe auch (5.6.1). In dem

Produkt auf der linken Seite von (5.6.4) fassen wir nun je p Faktoren zusammen: Für jedes $\ell \in \mathbb{N}_0$ sei $\widetilde{P}_\ell = P^{(\beta(n+\ell p+1))} P^{(\beta(n+\ell p+2))} \cdots P^{(\beta(n+(\ell+1)p))}$. Wenn $m > p$ und $k = \lfloor \frac{m}{p} \rfloor$, so haben wir

$$c\big(P^{(\beta(n+1))} P^{(\beta(n+2))} \cdots P^{(\beta(n+m))}\big) \leq c(\widetilde{P}_0) c(\widetilde{P}_1) \cdots c(\widetilde{P}_{k-1}).$$

Nun müssen wir also zeigen, dass $\lim_{k\to\infty} \prod_{\ell=1}^k c(\widetilde{P}_{\ell-1}) = 0$ ist. Dazu reicht es zu zeigen, dass $\sum_{\ell=1}^\infty (1 - c(\widetilde{P}_{\ell-1})) = \infty$ ist, denn es gilt $\log[\prod_{\ell=1}^k c(\widetilde{P}_{\ell-1})] = \sum_{\ell=1}^k \log(1 + (c(\widetilde{P}_{\ell-1}) - 1)) \leq \sum_{\ell=1}^k (c(\widetilde{P}_{\ell-1}) - 1)$. Wir setzen $K = \max_{i,j\in I}[H(j) - H(i)]^+$, wie in (5.6.1). Mit Hilfe von Lemma 5.6.3(a) sieht man, dass $1 - c(\widetilde{P}_{\ell-1}) \geq \min_{i,j\in I} \widetilde{P}_{\ell-1}(i,j)$. Also schätzen wir ab:

$$\begin{aligned}
\widetilde{P}_0(i,j) &= P^{(\beta(n+1))} P^{(\beta(n+2))} \cdots P^{(\beta(n+p))}(i,j) \\
&\geq \sum_{i_1,\dots,i_{p-1}\in I} q_{i,i_1} q_{i_1,i_2} \cdots q_{i_{p-2},i_{p-1}} q_{i_{p-1},j}\, e^{-\beta(n+1)[H(i_1)-H(i)]^+} e^{-\beta(n+2)[H(i_2)-H(i_1)]^+} \cdots \\
&\qquad\qquad \times e^{-\beta(n+p-1)[H(i_{p-1})-H(i_{p-2})]^+} e^{-\beta(n+p)[H(j)-H(i_{p-1})]^+} \\
&\geq q_{i,j}^{(p)} e^{-\beta(n+p)Kp} \\
&\geq \eta e^{-\beta(n+p)Kp}.
\end{aligned}$$

Analog schätzt man $\widetilde{P}_\ell(i,j)$ ab. Dies können wir einsetzen, um zu erhalten:

$$\sum_{\ell=1}^\infty (1 - c(\widetilde{P}_{\ell-1})) \geq \eta \sum_{\ell=1}^\infty e^{-\beta(n+\ell p)Kp}.$$

Nun endlich benutzen wir, dass $\beta(m) \leq \varepsilon \log m$ für jedes $m \in \mathbb{N}$ und erhalten

$$\sum_{\ell=1}^\infty (1 - c(\widetilde{P}_\ell)) \geq \eta \sum_{\ell=1}^\infty (n + \ell p)^{-\varepsilon Kp}.$$

Für genügend kleines $\varepsilon > 0$ ist diese Reihe divergent, genauer gesagt, für $\varepsilon < Kp$. Dies beendet den Beweis von (5.6.4). Dies setzen wir in (5.6.3) ein und lassen $n \to \infty$, wobei wir uns erinnern, dass $\lim_{n\to\infty} \delta_n = 0$. Daraus folgt (5.6.2), und der Beweis ist beendet. □

Satz 5.6.2 macht nur eine ziemlich grobe Aussage, und Abkühlung mit logarithmischer Geschwindigkeit ist recht langsam. Es gibt Untersuchungen, die schwächere Voraussetzungen machen, allerdings auf Basis einer genaueren Kenntnis der 'Landschaft', die die Höhen und Tiefen der Funktion H auf I erzeugen. Eine wichtige Größe, von denen solche Ergebnisse abhängen, ist z. B. die kleinste 'Passhöhe', die die Kette überwinden muss, um energetisch vorteilhafte Zustände zu erreichen. Doch sind diese genaueren Informationen meist nur sehr schwer zu erhalten, und selbst dann noch ist das Ergebnis für praktische Zwecke zu schlecht. Aus solchen Gründen benutzt man die Simulated-Annealing-Technik praktisch fast immer nur unter Verwendung von Fingerspitzengefühl und der Versuch-und-Irrtum-Methode.

Kapitel 6

Exaktes Ziehen von Stichproben

In diesem Kapitel stellen wir stochastische Algorithmen vor, die ein exaktes Ziehen einer Stichprobe auf einer endlichen Menge ermöglichen, d. h. die das im vorigen Kapitel gestellte Problem nicht approximativ, sondern exakt lösen. Diese Algorithmen weisen also nicht den Hauptnachteil der klassischen Markov-Chain-Monte-Carlo-Methoden auf, die wir in Kapitel 5 vorstellten: (1) Die gesuchte Verteilung wird in der Regel nicht erreicht, (2) die theoretische Rechtfertigung erfordert oft eine riesige Laufzeit, um mit Sicherheit davon ausgehen zu können, dass die zu simulierende Verteilung mit gegebener Genauigkeit erreicht ist (auch wenn die Praxis meist besser ausieht), (3) die optimale Wahl des Abbruchzeitpunkts erfordert entweder großen theoretischen Aufwand oder viel Fingerspitzengefühl.

Die in diesem Kapitel vorgestellten Algorithmen produzieren eine exakte Stichprobe, und sie geben auch ein klares Abbruchkriterium. Sie sind allerdings in der Regel nicht schneller, und sie verbrauchen auch potentiell enormen Speicherplatz. Ihre Grundlage ist – genau wie für die Algorithmen in Kapitel 5 – immer eine irreduzible und aperiodische Markovkette, deren invariante Verteilung die zu simulierende Verteilung ist. Doch statt diese Markovkette immer tiefer 'in die Zukunft hinein' zu iterieren, produziert man Stücke der Markovkette, die immer tiefer 'in der Vergangenheit' gestartet werden.

Ein wesentlicher Aspekt ist, dass in jedem Punkt des Zustandsraumes eine Kopie der Markovkette gestartet wird und dass die Sprungwahrscheinlichkeiten dieser Ketten mit einander auf eine geeignete Weise gekoppelt werden. In den meisten Anwendungen ist allerdings der Zustandsraum so gigantisch groß, dass eine Implementierung unmöglich ist. Daher sind in einigen Anwendungen (etwa dem Ising-Modell) modellabhängige Methoden entwickelt worden, die unter Ausnutzung gewisser Struktureigenschaften (wie etwa von Monotonie) die Anzahl der benötigten Ketten drastisch verringern, so dass eine Implementierung möglich wird.

Der Prototyp eines Algorithmus zur exakten Simulation, der *Propp-Wilson-Algorithmus*, wird in Abschnitt 6.1 vorgestellt und seine Korrektheit in 6.2 bewiesen. In 6.3 erläutern wir an Hand von Beispielen, wie gewisse Monotonieeigenschaften die Anzahl der benötigten Ketten drastisch verringern und somit die Implementierung erst ermöglichen. In 6.4 diskutieren wir eine Variante, die den Speicherplatzbedarf stark verringert. Zum Schluss stellen wir in Abschnitt 6.5 den *Fill-Algorithmus* vor, der es möglich macht, bei Ungeduld oder Zeitmangel den Algorithmus abzubrechen und neu zu starten.

6.1 Der Propp-Wilson-Algorithmus

Der Propp-Wilson-Algorithmus produziert eine exakte Stichprobe einer gegebenen Verteilung π auf einem gegebenen Zustandsraum I (den wir wieder als endlich annehmen wollen) mit Wahrscheinlichkeit Eins. Siehe [PW96] für die Originalarbeit und [Wi03] für eine Website, auf der D.B. Wilson eine enorme Materialfülle zum Problem des exakten Simulierens mit Markovketten zusammengetragen hat und permanent unterhält.

Wir nehmen an, dass es eine irreduzible und aperiodische stochastische Matrix P auf I gibt, deren invariante Verteilung π ist. Es ist für das Folgende sehr wesentlich, dass für die Simulation der Markovkette eine *Übergangsfunktion* $\phi\colon I \times [0,1] \to I$ existiert, so dass die Markovkette $(X_n)_{n\in\mathbb{N}_0}$ mit Übergangsmatrix P gebildet wird nach der *Übergangsregel*

$$X_{n+1} = \phi(X_n, U_n), \qquad n \in \mathbb{N}_0, \tag{6.1.1}$$

wobei $(U_n)_{n\in\mathbb{N}_0}$ eine Folge von unabhängigen, auf $[0,1]$ gleichförmig verteilten Zufallsvariablen ist (siehe Abschnitt 1.3). Um Trivialitäten zu vermeiden, gehen wir davon aus, dass die Treppenfunktionen $\phi(i,\cdot)$ nur nichttriviale Stufen haben.

Die intuitive Grundidee des Algorithmus ist die folgende. Es sei $(X_n)_{n\in-\mathbb{N}_0}$ eine Markovkette mit Übergangsmatrix P, die wir uns zum Zeitpunkt $-\infty$ gestartet denken und bis zum Zeitpunkt 0, der Gegenwart, betrachten. Da die Kette zum Zeitpunkt 0 ja schon unendlich viele Schritte gemacht hat, muss also X_0 schon die invariante Verteilung haben. Da wir aber praktisch nicht zum Zeitpunkt $-\infty$ starten können, wählen wir zunächst nur eine negative Zeit $-T_1 \in -\mathbb{N}$ und konstruieren für jeden Zustand $i \in I$ in dem Zustandsraum eine Markovkette $(X_n^{(i)})_{n=-T_1,\dots,0}$, die in i zum Zeitpunkt $-T_1$ startet und für T_1 Schritte konstruiert wird. Wichtig hierbei ist, dass wir alle diese Ketten mit Hilfe der selben Realisation $(U_n)_{n=-T_1,\dots,-1}$ der unabhängigen, auf $[0,1]$ gleichförmig verteilten Variablen über die Regel (6.1.1) konstruieren, also

$$X_{n+1}^{(i)} = \phi(X_n^{(i)}, U_n), \quad -T_1 \le n < 0, \qquad \text{mit} \qquad X_{-T_1}^{(i)} = i.$$

Dann hat die Familie dieser Ketten nämlich die *Verschmelzungseigenschaft*: Sobald je zwei von ihnen einmal zum selben Zeitpunkt im selben Zustand sind, bleiben sie danach immer zusammen und legen den selben Pfad zurück – man sagt, sie *verschmelzen* oder sie *koalieren*.

Stellen wir uns vor, dass zum Zeitpunkt 0 alle Ketten mit einander verschmolzen sind, also dass alle in einem einzigen Zustand sind. Betrachten wir eine zum Zeitpunkt $-\infty$ gestartete Kette $(X_n)_{n\in-\mathbb{N}_0}$, deren Übergangsentscheidungen zu den Zeitpunkten $n = -T_1, \dots, -1$ auf den oben gewählten $U_{-T_1}, \dots, U_{-1}$ basieren. Diese unendlich lange Kette muss in ihren letzten T_1 Schritten exakt in den Fußstapfen irgendeiner der Ketten $(X_n^{(i)})_{n=-T_1,\dots,0}$ verlaufen, nämlich in denen der Kette, die in X_{-T_1} gestartet wurde. Dann wäre sie also zum Zeitpunkt 0 ebenfalls in dem gemeinsamen Zustand, in dem alle diese Ketten verschmolzen sind. Daher sollte also dieser gemeinsame Zustand die perfekte Verteilung π haben, und wir haben das Problem gelöst.

Im anderen Fall, wo also nicht alle der Ketten $(X_n^{(i)})_{n=-T_1,\dots,0}$ mit einander verschmolzen sind, gehen wir tiefer in die Vergangenheit wie folgt. Mit einer Zeit $-T_2 < -T_1$ beschaffen wir uns weitere Realisationen $U_{-T_2}, \dots, U_{-T_1-1}$ von unabhängigen, auf $[0,1]$ gleichförmig verteilten Zufallsvariablen und konstruieren wiederum für jeden Zustand $i \in I$ je eine Markovkette, die zum Zeitpunkt $-T_2$ in i startet und bis zum Zeitpunkt $-T_1$ konstruiert wird. Dabei landen sie an den Startzuständen der oben konstruierten Ketten. Indem man die Ketten 'der zweiten Generation' jeweils verlängert um die im ersten Schritt konstruierte Kette, deren Startzustand gleich dem Endzustand der zum Zeitpunkt $-T_2$ gestarteten Kette ist, erhält man Ketten der Länge T_2, die also bis zum Zeitpunkt 0 betrachtet werden. (Diejenigen Ketten $(X_n^{(i)})_{n=-T_1,\dots,0}$, deren Startzustand von keiner der Ketten der zweiten Generation getroffen wird, können wir übrigens vergessen.) Wir haben also für jedes $i \in I$ eine Kette $(X_n^{(i)})_{n=-T_2,\dots,0}$ der Länge T_2, die in i startet, und die Familie dieser Ketten hat die Verschmelzungseigenschaft. Wiederum prüfen wir, ob alle diese Ketten zum Zeitpunkt 0 verschmolzen sind. Wenn ja, so ist dieser Verschmelzungszustand eine Stichprobe mit Verteilung π, falls nein, so müssen wir wiederum tiefer in die Vergangenheit gehen.

Wegen seines Charakters des sukzessiven tieferen Eindringens in die Vergangenheit und der gekoppelten Betrachtung mehrerer Markovketten nennt man diesen Algorithmus auch *Coupling from the past*. Das Abbruchkriterium des Algorithmus liegt auf der Hand: Wenn alle betrachteten Ketten zum Zeitpunkt 0 mit einander verschmolzen sind, ist der Algorithmus beendet.

Propp und Wilson zeigten in [PW96], dass die Wahl $T_k = 2^{k-1}$ zu einem nahezu optimalen Verlauf führt, siehe auch Bemerkung 6.2.3. Die Tasache, ob und wann die Verschmelzung erreicht wird, hängt zwar nicht von der Wahl der T_k ab, aber die Anzahl der Vergleiche, die nötig sind, um die Verschmelzung zu erkennen.

6.2 Korrektheit des Propp-Wilson-Algorithmus

Um die Korrektheit des Propp-Wilson-Algorithmus zu beweisen, müssen nach einander zwei Dinge bewiesen werden: (1) Der Algorithmus terminiert in endlicher Zeit (d. h. der Verschmelzungszeitpunkt ist eine fast sicher endliche Zufallsgröße), (2) zum Zeitpunkt des Terminierung hat die Ausgabe die korrekte Verteilung. Es stellt sich heraus, dass die Tatsache, ob der Algorithmus terminiert oder nicht, ausschließlich von der Wahl der Übergangsfunktion ϕ in (6.1.1) abhängt, so lange eine aperiodische irreduzible Markovkette verwendet wird.[1]

Die iterative Anwendung der Regel (6.1.1) für die einzelnen Schritte der jeweiligen Markovketten führt zu der Iteration zufälliger Funktionen, denn es gilt für die in i gestartete Markovkette der Länge T_1:

$$X_0^{(i)} = \phi(\phi(\dots\phi(\phi(i, U_{-T_1}), U_{-T_1+1})\dots, U_2), U_1). \tag{6.2.1}$$

Wir führen folgende Notation ein:

$$\phi^{(N)}\big(i, (u)_{-N}^{-1}\big) = \phi(\phi(\dots\phi(\phi(i, u_{-N}), u_{-N+1})\dots, u_2), u_1), \qquad N \in \mathbb{N}, u_{-N}, \dots, u_{-1} \in [0,1], \tag{6.2.2}$$

dann ist $\phi^{(N)}(\,\cdot\,, (u)_{-N}^{-1}) \in I^I$ eine Abbildung $I \to I$. Die Mindestforderung an ϕ, damit der Algorithmus eine Chance hat zu terminieren, ist also, dass die Iterationskette mit positiver Wahrscheinlichkeit zur Verschmelzung kommen kann, d. h. dass $\phi^{(N)}(\,\cdot\,, (u)_{-N}^{-1})$ für geeignete N und $u_{-N}, \dots, u_{-1}$ in der Menge M aller konstanten Abbildungen liegt. Es stellt sich heraus, dass diesem Fall der Verschmelzungszeitpunkt sogar endliche Erwartung hat:

Satz 6.2.1 (0-1-Gesetz für Termination). *Falls $N^* \in \mathbb{N}$, $x \in I$ und $u_{-N^*}, \dots, u_{-1} \in [0,1]$ existieren mit $\phi^{(N^*)}(i, (u)_{-N^*}^{-1}) = x$ für alle $i \in I$, dann terminiert der Propp-Wilson-Algorithmus mit Wahrscheinlichkeit 1, und der Verschmelzungszeitpunkt hat endlichen Erwartungswert. Anderen Falls terminiert er nie.*

[1] Siehe Bemerkung 6.2.6 für eine ungeschickte Wahl, so dass der Verschmelzungszeitpunkt nie erreicht wird.

Beweis. Es ist klar, dass der Algorithmus nie terminieren kann, wenn keine Realisationen existieren, die dies bewerkstelligen; der zweite Teil des Satzes ist also trivial.

Wir betrachten den Vektor $\Phi_1 = (\phi^{(N^*)}(i, (U)_{-N^*}^{-1}))_{i\in I} \in I^I$, der die Position aller Ketten $X^{(i)}$ zum Zeitpunkt 0 angibt, wenn man den Algorithmus zum Zeitpunkt $-N^*$ gestartet hat. Da wir voraussetzten, dass die Treppenfunktionen $\phi(i,\cdot)$ nur nichttriviale Stufen haben (siehe den Text nach (6.1.1)), bedeutet die Voraussetzung des Satzes, dass ein $\varepsilon > 0$ existiert mit $\mathbb{P}(\Phi_1 \in M) > \varepsilon$, wobei M die Menge der konstanten Abbildungen $I \to I$ bezeichnet. Die Wahrscheinlichkeit, dass der Verschmelzungszeitpunkt τ der Propp-Wilson-Kette größer als N^* ist, ist also $\leq 1 - \varepsilon$. Die Abbildungen

$$\Phi_\ell = (\phi^{(N^*)}(i, (U)_{-\ell N^*}^{-(\ell-1)N^*-1})_{i\in I} \in I^I, \qquad \ell \in \mathbb{N},$$

sind unabhängige Kopien von Φ_1. Sei

$$L^* = \inf\{\ell \in \mathbb{N} \colon \Phi_\ell \in M\}$$

das kleinste $\ell \in \mathbb{N}$, so dass Φ_ℓ die Verschmelzung erreicht. Dann ist L^* nach oben beschränkt durch eine geometrisch zum Parameter ε verteilte Zufallsgröße, besitzt also einen endlichen Erwartungswert. Wenn man die ersten L^* Ketten in einander einsetzt, erhält man die Abbildung $\Phi_1 \circ \Phi_2 \circ \cdots \circ \Phi_{L^*} = (\phi^{(L^*N^*)}(i, (U)_{-L^*N^*}^{-1}))_{i\in I}$, die in M liegt, also verschmolzen ist. Wegen $\tau \leq N^* L^*$ hat also auch τ einen endlichen Erwartungswert. Wenn man k so groß wählt, dass $-T_k \leq -L^* N^*$, so terminiert der Propp-Wilson-Algorithmus also nach dem k-ten Schritt. $\square$

Wir bringen nun einen Beweis für die Korrektheit der Verteilung der Ausgabe des Algorithmus unter der Voraussetzung der Termination.

Satz 6.2.2. *Es sei $T_k = 2^{k-1}$ für $k \in \mathbb{N}$ gewählt, und es sei vorausgesetzt, dass der Propp-Wilson-Algorithmus terminiert. Sei Y der Verschmelzungszustand der Ketten. Dann gilt $\mathbb{P}(Y = i) = \pi(i)$ für jedes $i \in I$.*

Beweis. Es sei A_k das Ereignis, dass der Algorithmus mindestens bis zur Vergangenheit $-T_k$ zurück gehen muss, um die Verschmelzung aller zum Zeitpunkt $-T_k$ gestarteten Ketten zum Zeitpunkt Null zu erreichen. Die Ereignisse A_k sind ineinander aufsteigend enthalten, und auf Grund der Voraussetzung der Termination gilt $\lim_{k\to\infty} \mathbb{P}(A_k) = \mathbb{P}(\bigcup_{k\in\mathbb{N}} A_k) = 1$. Sei $\varepsilon > 0$ vorgegeben, dann finden wir also ein $k \in \mathbb{N}$ mit $\mathbb{P}(A_k) \geq 1 - \varepsilon$.

Nun betrachten wir eine ‘perfekte’ Markovkette $(Y_n)_{n=-T_k,\dots,0}$, die mit der invarianten Verteilung π zum Zeitpunkt $-T_k$ gestartet wird und daher zu jedem Zeitpunkt die Verteilung π exakt hat. Wir nehmen an, dass sie die Regel (6.1.1) benutzt, und zwar mit den selben Realisationen $U_{-T_k}, \dots, U_{-1}$, die auch die Ketten des Algorithmus benutzen. Wegen der resultierenden Verschmelzungseigenschaft gilt also $Y = Y(0)$ auf A_k und daher insbesondere $\mathbb{P}(Y = Y(0)) \geq 1 - \varepsilon$. Wegen

$$\mathbb{P}(Y = i) - \pi(i) = \mathbb{P}(Y = i) - \mathbb{P}(Y(0) = i) \leq \mathbb{P}(Y = i, Y(0) \neq i) \leq \mathbb{P}(Y \neq Y(0)) \leq \varepsilon$$

und analog

$$\pi(i) - \mathbb{P}(Y = i) = \mathbb{P}(Y(0) = i) - \mathbb{P}(Y = i) \leq \mathbb{P}(Y(0) = i, Y \neq i) \leq \mathbb{P}(Y \neq Y(0)) \leq \varepsilon$$

folgt $|\mathbb{P}(Y = i) - \pi(i)| \leq \varepsilon$. Da $\varepsilon > 0$ beliebig war, ist der Beweis beendet. $\square$

Bemerkung 6.2.3 (Aufwand). Die Wahl der T_k hat keinen Einfluss auf den Verschmelzungszeitpunkt, aber auf die Gesamtanzahl der Iterationen der zufälligen Funktionen $\phi(\,\cdot\,,u)$, d. h. auf die Gesamtanzahl der Markovschen Schritte. Wenn man $T_k = k$ wählt, so müssen $1+2+3+\dots+K^* = \frac{1}{2}K^*(K^*+1)$ Markovschritte ausgeführt werden, wobei K^* die kleinste Zahl k ist, so dass die zum Zeitpunkt $-k$ gestarteten Ketten zum Zeitpunkt Null verschmolzen sind. Wenn allerdings $T_k = 2^{k-1}$ gewählt wird, dann ist die Zahl der Markovschritte gleich $1+2+4+8+\dots+2^{L^*} = 2^{L^*+1}-1$, wobei $L^* \in \mathbb{N}$ so gewählt ist, dass $2^{L^*-1} < K^* \le 2^{L^*}$. Also ist diese Anzahl nicht größer als $4K^*$, d. h. diese Wahl der T_k ist effizienter. ◇

Bemerkung 6.2.4 ('Coupling to the future'). Eine auf der Hand liegende Variante des Propp-Wilson-Algorithmus ist die folgende: Wir starten in jedem Zustand eine Markovkette zum Zeitpunkt 0 und lassen sie alle mit den selben Übergangsregeln so lange laufen, bis sie alle in einem Zustand verschmolzen sind. Dann müsste doch dieser Verschmelzungszustand die gesuchte Verteilung haben.

Dies ist allerdings nicht richtig, wie das Gegenbeispiel

$$\pi = \left(\tfrac{2}{3}, \tfrac{1}{3}\right), \qquad P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 1 & 0 \end{pmatrix} \tag{6.2.3}$$

zeigt. Wie man leicht sieht, ist π die invariante Verteilung von P auf dem Zustandsraum $I = \{1,2\}$. Lassen wir also zwei Markovketten, die wir in 1 bzw. in 2 starten, laufen, bis sie zum Zeitpunkt N erstmals miteinander verschmelzen. Daher sind sie zum Zeitpunkt $N-1$ in verschiedenen Zuständen. Daher können die Ketten zum Zeitpunkt N nicht in 2 sein, denn von 2 kann man nicht zu 2 in einem Sprung gelangen, aber eine dieser Ketten kam ja von diesem Zustand. Daraus folgt, dass sich die Ketten nur in 1 zum ersten Mal treffen können, d. h. die Verteilung zum Verschmelzungszeitpunkt ist konzentriert in 1. Dies ist natürlich nicht die invariante Verteilung π. ◇

Bemerkung 6.2.5 (Frisches Ziehen). Es ist für die Korrektheit des Propp-Wilson-Algorithmus wesentlich, für alle betrachteten Ketten ab dem Zeitpunkt $-T_k$ die selben Realisationen $U_{-T_k}, \dots, U_{-1}$ zu benutzen. Wenn wir etwa für jede Kettenfamilie, die zum Zeitpunkt $-T_k$ gestartet wird, 'neu gezogene' Realisationen der unabhängigen, gleichförmig auf $[0,1]$ verteilten Zufallsvariablen U_n mit $n = -T_k, \dots, -1$, also für die gesamte Lauflänge, benutzten (d. h. für das Intervall $\{-T_{k-1}, \dots, -1\}$ nicht die Sprungentscheidungen der zum Zeitpunkt $-T_{k-1}$ gestarteten Ketten benutzten), so erhalten wir zum Verschmelzungszeitpunkt nicht die gewünschte Verteilung. Als Beispiel betrachten wir die Kette in (6.2.3). Mit der Wahl $T_k = 2^{k-1}$ etwa betrachten wir den (zufälligen) kleinsten Index $K \in \mathbb{N}$, so dass die zum Zeitpunkt $-T_K$ gestarteten Ketten zum Zeitpunkt 0 verschmelzen, dann gilt:

$$\begin{aligned}
\mathbb{P}(Y=1) &= \sum_{k=1}^{\infty} \mathbb{P}(K=k, Y=1) \\
&\ge \mathbb{P}(K=1, Y=1) + \mathbb{P}(K=2, Y=1) \\
&= \mathbb{P}(K=1)\mathbb{P}(Y=1 \mid K=1) + \mathbb{P}(K=2)\mathbb{P}(Y=1 \mid K=2) \\
&= \frac{1}{2}\cdot 1 + \frac{3}{8}\cdot\frac{2}{3} = \frac{3}{4} > \frac{2}{3} = \pi(1).
\end{aligned}$$

(Die drittletzte Gleichung sieht man so ein: $\mathbb{P}(K=1) = \frac{1}{2}$ ist klar, und auf $\{K=1\}$ muss offensichtlich auch $Y=1$ sein. Um das Ereignis $\{K>1\}$ zu realisieren, müssen zunächst ein

Auf- und ein Absprung gleichzeitig passiert sein, das hat die Wahrscheinlichkeit $\frac{1}{2}$. Um $\{K = 2\}$ gegeben $\{K > 1\}$ zu realisieren (nun werden ja beide Sprungentscheidungen zu den Zeitpunkten -2 und -1 neu ausgewürfelt), müssen genau drei der vier möglichen Sprungkombinationen geschehen, und dies hat die Wahrscheinlichkeit $\frac{3}{4}$. Dies zeigt, dass $\mathbb{P}(K = 2) = \frac{3}{8}$. In zwei der drei Möglichkeiten sind die beiden Ketten in 1 zum Zeitpunkt Null, also gilt $\mathbb{P}(Y = 1 \mid K = 2) = \frac{2}{3}$.)

Also hat Y dann eine andere Verteilung als π.

Die Idee des ständigen Neu-Auswürfelns der Zufallsvariablen $U_{-T_k}, \dots, U_{-1}$ wird aber in Abschnitt 6.4 verfolgt und ausgebaut werden, denn sie verringert den Nachteil des Propp-Wilson-Algorithmus des großen Speicherplatzverbrauchs. $\diamond$

Bemerkung 6.2.6 (Einfluss der Übergangsfunktion). Das folgende Beispiel zeigt, dass die Wahl der Übergangsfunktion ϕ in (6.1.1) eine wichtige Rolle spielt für die Termination des Algorithmus. Betrachten wir $I = \{1, 2\}$ und die Übergangsmatrix

$$\pi = \left(\tfrac{1}{2}, \tfrac{1}{2}\right), \qquad P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}. \tag{6.2.4}$$

Die zugehörige Markovkette ist eine Folge unabhängiger, auf I gleichförmig verteilter Zufallsvariablen, es handelt sich also um ein sehr simples Beispiel. Als Übergangsfunktion kann man wählen:

$$\phi(i, u) = \begin{cases} 1, & \text{falls } u \in [0, \frac{1}{2}], \\ 2, & \text{falls } u \in (\frac{1}{2}, 1], \end{cases} \tag{6.2.5}$$

aber auch das durch

$$\phi(1, u) = \begin{cases} 1, & \text{falls } u \in [0, \frac{1}{2}], \\ 2, & \text{falls } u \in (\frac{1}{2}, 1], \end{cases} \qquad \text{und} \qquad \phi(2, u) = \begin{cases} 2, & \text{falls } u \in [0, \frac{1}{2}], \\ 1, & \text{falls } u \in (\frac{1}{2}, 1], \end{cases} \tag{6.2.6}$$

gegebene ϕ. Man kann leicht zeigen, dass die Wahl in (6.2.5) dazu führt, dass mit der Wahl $T_1 = 1$ die zum Zeitpunkt $-T_1$ gestarteten Ketten mit Wahrscheinlichkeit Eins zum Zeitpunkt 0 verschmelzen, aber die Wahl in (6.2.6) impliziert, dass die Ketten sich mit Wahrscheinlichkeit Eins gegenseitig nie treffen, egal wie tief in die Vergangenheit geschaut wird. $\diamond$

6.3 Monotonie und 'Sandwichen'

Wenn der Zustandsraum I sehr groß ist, dann ist es unmöglich, *jede* der in den einzelnen Zuständen gestartete Markovketten zu implementieren, und der Algorithmus hat nur theoretischen Wert. In vielen konkreten Anwendungen gibt es allerdings Möglichkeiten, gewisse Struktureigenschaften auszunutzen, so dass die Anzahl der benötigten Ketten drastisch verringert wird. (Nebenbei wird auch noch der Aufwand des Testens, ob alle beteiligten Ketten mit einander verschmolzen sind, stark verkleinert.) In diesem Abschnitt diskutieren wir die Monotonie als eine solche Möglichkeit.

Die Idee ist die folgende. Falls auf dem Zustandsraum I eine Ordnung existiert und ein kleinstes Element $\widehat{0}$ und ein größtes $\widehat{1}$ und wenn man die Ketten so konstruieren kann, dass die anfängliche Ordnung dieser Ketten in jedem Markovschritt beibehalten wird, dann haben wir Verschmelzung genau dann, wenn die in $\widehat{0}$ gestartete Kette mit der in $\widehat{1}$ gestarteten verschmolzen ist.[2] Also müssen wir in diesem Fall tatsächlich nur zwei Ketten laufen lassen, was den Aufwand enorm verringert.

Die Eigenschaft der Ordnungserhaltung hängt nicht nur von der Markovkette P ab, sondern auch von der Wahl der Übergangsregel, wie wir im folgenden simplen Beispiel demonstrieren.

Beispiel 6.3.1 (Irrfahrten). Wir betrachten auf $I = \{1, \dots, k\}$ die symmetrische Irrfahrt, also $p_{i,i+1} = p_{i,i-1} = \frac{1}{2}$ für $i = 2, \dots, k-1$ mit Randbedingungen $p_{1,1} = p_{1,2} = p_{k,k} = p_{k,k-1} = \frac{1}{2}$. Die invariante Verteilung π ist die Gleichverteilung auf I. Wir wählen die Übergangsfunktion mit

$$\phi(1,u) = \begin{cases} 1, & \text{falls } u \in [0, \frac{1}{2}], \\ 2 & \text{sonst,} \end{cases} \qquad \text{und} \qquad \phi(k,u) = \begin{cases} k-1, & \text{falls } u \in [0, \frac{1}{2}], \\ k & \text{sonst,} \end{cases}$$

und

$$\phi(i,u) = \begin{cases} i-1, & \text{falls } u \in [0, \frac{1}{2}], \\ i+1 & \text{sonst.} \end{cases}$$

Die mit diesem ϕ nach (6.1.1) konstruierte Markovkette funktioniert also nach dem Motto: ‘Wenn $u \in [0, \frac{1}{2}]$, so wähle die kleinere Möglichkeit, im Fall $u \in (\frac{1}{2}, 1]$ die größere’. Da alle Ketten diesem Motto folgen, wird die Ordnung der k in den Zuständen $1, \dots, k$ gestarteten Ketten also zu keinem Zeitpunkt verändert. Die Eigenschaft von ϕ, die hierzu führte, ist die folgende:

$$\text{Für jedes } u \in [0,1] \text{ gilt:} \qquad i \le j \quad \Longrightarrow \quad \phi(i,u) \le \phi(j,u),$$

d. h. $\phi(\cdot, u)$ ist monoton steigend für jedes $u \in [0,1]$.

Es ist leicht, eine andere Übergangsfunktion anzugeben, die es ermöglicht, dass zwei Ketten mit einander die Plätze wechseln, also die Ordnung verändern. ◇

Auch im Ising-Modell (siehe Beispiele 5.1.4 und 5.3.2) gibt es eine geeignete Ordnungsstruktur. Es sei bemerkt, dass es die Anwendung auf dieses Modell war, die im Originalartikel [PW96] ausführlich diskutiert wurde und auf diese Weise sehr schnell die Aufmerksamkeit auf sich zog, so dass der Propp-Wilson-Algorithmus schnell bekannt und anerkannt wurde.

Beispiel 6.3.2 (Ising-Modell). Wir betrachten das Ising-Modell auf $I = \{-1, 1\}^V$ mit einem beliebigen endlichen Grafen $G = (V, E)$. Auf dem Zustandsraum I haben wir die Halbordnung

$$i \le j \qquad \Longleftrightarrow \qquad \forall x \in V : i_x \le j_x$$

für Konfigurationen $i = (i_x)_{x \in V}$ und $j = (j_x)_{x \in V}$. Diese Ordnung ist zwar nicht total, d. h. nicht jede zwei Konfigurationen können mit einander verglichen werden, aber es gibt genau eine größte Konfiguration $\widehat{1}$ und eine kleinste $\widehat{0}$, die komplett aus Einsen bzw. aus -1 besteht. Jede Konfiguration i erfüllt $\widehat{0} \le i \le \widehat{1}$.

Als zu Grunde liegende Markovkette benutzen wir den Gibbs-Sampler aus Abschnitt 5.4. Sei also $i = (i_x)_{x \in V}$ eine Konfiguration. Wir müssen nun eine Übergangsfunktion $\phi(i, u)$ für die Gibbs-Sampler-Markovkette so festlegen, so dass $\phi(\cdot, u)$ für jedes $u \in [0,1]$ monoton steigend ist. Hierzu formulieren wir ein zweistufiges Programm, d. h. wir führen die Markovkette mit Hilfe von zwei unabhängigen auf $[0,1]$ gleichförmig verteilten Zufallsvariablen aus, indem wir die erste für die Wahl des Gitterpunktes benutzen (Schritt (i) im Algorithmus) und die zweite für die Wahl des Spins in diesem Gitterpunkt (Schritt (ii)). Mit der Übergangsfunktion $\phi_1(i, u) = x$, falls

[2] Im Fachjargon redet man auch von ‘Sandwichen’.

$u \in [\frac{x-1}{|V|}, \frac{x}{|V|})$, ‘würfeln’ wir den Gitterpunkt $x \in V = \{1, \dots, |V|\}$ mit der Gleichverteilung aus, und mit der Funktion

$$\phi_2(i, x, u) = \begin{cases} +1, & \text{falls } u \le \left[e^{-4\beta[k_+(i,x)-k_-(i,x)]}\right]^{-1}, \\ -1 & \text{sonst}, \end{cases}$$

wählen wir den Spin der Konfiguration i im Gitterpunkt x nach der bedingten Verteilung gegeben die Spins der Nachbarn von x; hierbei ist $k_+(i, x)$ die Anzahl der $+1$-Spins der Nachbarpunkte von x in i, und $k_-(i, x)$ die Anzahl der -1-Spins; siehe Beispiel 5.3.2. Es ist leicht zu sehen, dass für jedes $u \in [0, 1]$ und für jedes $x \in V$ die Funktion $\phi_2(\cdot, x, u)$ monoton steigend bezüglich der oben definierten Halbordnung ist, denn für $i \le j$ ist $k_+(i, x) \le k_+(j, x)$ und $k_-(i, x) \ge k_-(j, x)$ für jedes $x \in V$. Also hat die mit dieser Übergangsfunktion konstruierte Markovkette die Monotonieeigenschaft, die es ermöglicht, die Verschmelzung aller Ketten allein mit der Verschmelzung der in $\widehat{1}$ und $\widehat{0}$ gestarteten Ketten zu prüfen. $\diamond$

Wir beschränken uns auf diese beiden Beispiele. Im Grafenfärbungsmodell in Beispiel 5.5.2 gibt es keine Monotonieeigenschaft, die ein analoges Vorgehen ermöglicht, aber es sind alternative Methoden entwickelt worden.

6.4 Wilsons Modifikation

Der Propp-Wilson-Algorithmus weist den Nachteil eines hohen Speicherplatzbedarfs auf, denn die Realisationen der unabhängigen auf $[0, 1]$ gleichförmig verteilten Zufallsvariablen U_n müssen ja bei jedem Markovschritt einer Kette wieder benutzt werden, ohne sie neu auszuwürfeln: Wir hatten in Bemerkung 6.2.5 gesehen, dass eine unabhängige Neuauswürflung zu einer falschen Verteilung des Verschmelzungszustands führt. Im Extremfall könnte also der Algorithmus mitten in der Ausführung zum Abbruch gezwungen sein, wenn der Speicherplatz nicht mehr ausreicht.

Um diesem Problem zu begegnen, entwarf Wilson [Wi00] eine Modifikation des Algorithmus, in der für jeden Markovschritt neue Realisationen benutzt werden können, so dass also der Speicherplatzbedarf drastisch sinkt. Ferner hat die Modifikation die Eigenschaft, dass sie dem Prinzip *Coupling to the future* folgt, also die Markovketten immer tiefer in die Zukunft hinein konstruiert. In Beispiel 6.2.4 sahen wir, dass dann der Verschmelzungszustand ebenfalls eine falsche Verteilung haben kann, also muss hier modifiziert werden. Dies wird gemacht, indem nach dem Verschmelzungszeitpunkt die Markovketten weitergeführt werden bis zu einem zufälligen Zeitpunkt, dessen Definition relativ kompliziert ist.

Um Trivialitäten zu vermeiden, setzen wir voraus, dass die Übergangsregel ϕ in (6.1.1) so gewählt wird, dass die Termination mit Wahrscheinlichkeit Eins geschieht (vergleiche Satz 6.2.1). Es sei $(\tau_k)_{k\in\mathbb{N}}$ eine Folge von unabhängigen identisch verteilten $\mathbb{N}$-wertigen Zufallsgrößen, die jeweils die Verteilung der Verschmelzungszeit einer Propp-Wilson-Familie von Markovketten ist, also der erste Zeitpunkt, zu dem alle in den jeweiligen Zuständen des Zustandsraums I gestarteten Ketten sich in einem einzigen Zustand getroffen haben. Eine Realisation der Zufallsgröße τ_k erhält man, indem man dieses Verschmelzungsexperiment mit Hilfsketten ausführt. Wir lassen also die k-te unabhängige Realisation der Markovkettenfamilie laufen, bis sie verschmilzt; dies passiert zum Zeitpunkt τ_k. Ein wenig formaler kann dies beschrieben werden mit einer Familie $(U_n^{(k)})_{n\in-\mathbb{N},k\in\mathbb{N}}$ von unabhängigen, auf $[0, 1]$ gleichförmig verteilten Zufallsgrößen. Dann betrachten wir nämlich die Folge $\Phi_n^{(k)} = \phi^{(n)}(\,\cdot\,, (U^{(k)})_{-n}^{-1})$, die wir bis $\tau_k = \inf\{n \in \mathbb{N} \colon \Phi_n^{(k)} \in M\}$ laufen lassen, wobei M die Menge aller konstanten Abbildungen $I \to I$ bezeichnet.

Nun wählen wir die Zeiten $-T_1, -T_2, \dots$ des Propp-Wilson-Algorithmus *zufällig* nach der Vorschrift

$$T_k = \sum_{j=1}^{k} \tau_j, \qquad k \in \mathbb{N}_0. \tag{6.4.1}$$

Wir wählen die Folge $(U_n)_{n\in-\mathbb{N}}$, die in der Konstruktion der Propp-Wilson-Ketten benutzt werden, unabhängig von der Familie $(U_n^{(k)})_{n\in-\mathbb{N},k\in\mathbb{N}}$, also ist auch die Folge der Zeiten τ_k unabhängig von der Folge $(U_n)_{n\in-\mathbb{N}}$. Der Beweis der Korrektheit in Satz 6.2.2 zeigt, dass der Algorithmus auch mit den so definierten zufälligen T_k korrekt funktioniert.

Damit ist der modifizierte Propp-Wilson-Algorithmus schon vollständig definiert. Nun beleuchten wir diese Modifikation aus anderem Winkel und erläutern, dass er implementiert werden kann als ein *Coupling to the future* Algorithmus und dass keine der auf $[0,1]$ gleichförmig verteilten Variablen zweimal ausgewertet wird.

Der Vektor

$$\widetilde{\Phi}^{(k)}_{\tau_k} = \phi^{(\tau_k)}(\,\cdot\,, (U)_{-T_k}^{-T_{k-1}-1})$$

beschreibt den Propp-Wilson-Prozess von Markovketten im Zeitintervall $\{-T_k, \dots, -T_{k-1}\}$. Die Folgen $(\widetilde{\Phi}^{(k)}_n)_{n\in\mathbb{N}}$ und $(\Phi^{(k)}_n)_{n\in\mathbb{N}}$ von Hilfsketten sind unabhängige Kopien von einander, und als Zeitdauer des ersten Prozess wählen wir die Verschmelzungszeiten des zweiten. Es ist keineswegs gewährleistet, dass die Ketten, die zum Zeitpunkt $-T_k$ gestartet wurden, bis zum Zeitpunkt $-T_{k-1}$ verschmolzen sind (d. h. dass der Vektor $\widetilde{\Phi}^{(k)}_{\tau_k}$ eine konstante Abbildung ist), denn die Verschmelzungszeit

$$\sigma_k = \inf\{n \in \mathbb{N} \colon \widetilde{\Phi}^{(k)}_n \in M\}$$

ist ja unabhängig von der gewählten Laufzeit τ_k. Man beachte, dass die Verschmelzung dieses Prozesses genau das Ereignis $\{\sigma_k \le \tau_k\}$ ist. Ferner beachte man, dass die Folge der Größen σ_k und τ_k eine unabhängige, identisch verteilte Folge ist. Insbesondere ist

$$p = \mathbb{P}(\sigma_k \le \tau_k) = \mathbb{P}(\sigma_k \ge \tau_k),$$

und die Summe dieser beiden Wahrscheinlichkeiten ist mindestens 1 (denn sie ist gleich der Summe der Wahrscheinlichkeiten von $\{\sigma_k > \tau_k\}$ und $\{\sigma_k < \tau_k\}$ und zwei Mal der von $\{\sigma_k = \tau_k\}$). Also gilt $p \ge \frac{1}{2}$. Daher ist die Folge der Ereignisse, dass die zum Zeitpunkt $-T_k$ gestartete Kettenfamilie bis zum Zeitpunkt $-T_{k-1}$ verschmolzen sind, eine Folge unabhängiger identisch verteilter Ereignisse mit Wahrscheinlichkeit p. Setze

$$J = \inf\{k \in \mathbb{N} \colon \widetilde{\Phi}^{(k)}_{\tau_k} \in M\},$$

dann ist J geometrisch zum Parameter p verteilt. Für $k = 1, \dots, J-1$ verschmelzen also die zum Zeitpunkt $-T_k$ gestartete Kettenfamilie bis zum Zeitpunkt $-T_{k-1}$ nicht, aber die J-te Generation tut dies. Da der Wert von p im Allgemeinen unbekannt ist, arbeitet man statt dessen mit dem Parameter $\frac{1}{2}$, indem man auf dem Ereignis $\{\sigma_k = \tau_k\}$ eine unabhängige faire Münze wirft und sich in diesem Fall mit Wahrscheinlichkeit $\frac{1}{2}$ für eine der beiden Kettenfamilien entscheidet.

Also kann Wilsons Modifikation auf die folgende Weise implementiert werden: Wir generieren zunächst eine geometrisch verteilte Zufallgröße J mit Parameter $\frac{1}{2}$. Da die Folgen $(\widetilde{\Phi}^{(k)}_n)_{n\in\mathbb{N}}$ und $(\Phi^{(k)}_n)_{n\in\mathbb{N}}$ unabhängige Kopien von einander sind, kann man die Läufe dieser beiden Prozesse als einen 'Zwillingslauf' betrachten, von denen wir viele unabhängige Kopien laufen lassen, bis eine von ihnen verschmilzt. Die erste Kopie macht dies zum Zeitpunkt σ_k, die zweite zum Zeitpunkt τ_k, und wir betrachten beide Kopien bis zum Zeitpunkt $\sigma_k \wedge \tau_k$. Bei den ersten $J-1$

dieser Zwillingsläufe wählen wir den 'Verlierer' aus (d. h. diejenige Kette, die bis zum Zeitpunkt $\sigma_k \wedge \tau_k$ noch nicht verschmolzen ist, bzw., falls beide verschmolzen sind, eine zufällige von ihnen), aber beim J-ten Mal den Gewinner. Wenn wir diese Kopien an einander hängen, erhalten wir eine Realisation des oben definierten Propp-Wilson-Algorithmus mit den in (6.4.1) festgelegten zufälligen T_k. Bei allen Markovsprüngen werden hierbei neu ausgewürfelte Realisationen der benutzten Zufallsvariablen U_n bzw. $U_n^{(k)}$ benutzt, so dass keine dieser Werte gespeichert werden muss, und dies war die ursprüngliche Motivation.

6.5 Der Fill-Algorithmus

Einer der Nachteile des Propp-Wilson-Algorithmus ist die Tatsache, dass der Verschmelzungszeitpunkt und die Ausgabe des Algorithmus nicht unabhängig sind, sondern stark korrelieren. Wenn der Anwender also aus Ungeduld oder Zeitmangel den Algorithmus abbrechen möchte, um ihn neu zu starten, so muss er ganz am Anfang beginnen und kann nicht den Zustand zum Abbruchzeitpunkt verwenden. Der *Fill-Algorithmus*, den wir in diesem Abschnitt vorstellen, vermeidet diesen Nachteil, indem er die gesuchte Verteilung unabhängig von der Anzahl der Iterationsschritte generiert. Daher kann der Algorithmus ohne Probleme unterbrochen und neu gestartet werden.

Wir legen wieder eine Markovkette mit irreduzibler aperiodischer Übergangsmatrix P zu Grunde, die die gesuchte Verteilung π auf I als invariante Verteilung besitzt. Wir benötigen auch die zeitinverse Markovkette, die die Übergangsmatrix $\widetilde{P} = (\widetilde{p}_{i,j})_{i,j\in I}$ besitzt, die gegeben ist durch

$$\widetilde{p}_{i,j} = \frac{\pi(j)p_{j,i}}{\pi(i)}, \qquad i,j \in I. \tag{6.5.1}$$

Falls also die Kette $(X_0,\dots,X_k)$ die Übergangsmatrix P hat, so hat $(X_k,\dots,X_0)$ die Übergangsmatrix $\widetilde{P}$. Auch $\widetilde{P}$ ist ergodisch und besitzt π als invariante Verteilung.

Der Algorithmus von Fill setzt nicht voraus, dass für $\widetilde{P}$ eine monotone Übergangsregel $\widetilde{\phi}$ wie in (6.1.1) existiert, aber wir tun dies hier zur Vereinfachung, d. h. wir behandeln nur die Version des Fillschen Algorithmus bei Vorliegen und unter Ausnutzung von Monotonie, wie in Abschnitt 6.3. Auf I gebe es also eine Halbordnung $\leq$ und eindeutig bestimmte extremale Elemente $\widehat{0}$ und $\widehat{1}$, so dass $\widehat{0} \leq i \leq \widehat{1}$ für jedes $i \in I$, und es gebe eine Übergangsregel $\widetilde{\phi}$, so dass $\widetilde{\phi}(\,\cdot\,,u)$ monoton steigend ist für jedes $u \in [0,1]$.

Der Algorithmus von Fill ist folgender Maßen definiert. Für $k = 1,2,4,8,\dots$ führt man unabhängig die folgende Routine aus, bis zum ersten Mal eine Ausgabe erhalten wird. Zunächst produzieren wir eine Markovkette $(X_0,\dots,X_k)$ mit Übergangsmatrix P, die in $X_0 = \widehat{0}$ gestartet wird, und wir setzen $z = X_k$. Wir setzen $\widetilde{X} = (\widetilde{X}_0,\dots,\widetilde{X}_k) = (X_k,\dots,X_0)$, dann hat diese Kette die Übergangsmatrix $\widetilde{P}$. Gegeben den Pfad $\widetilde{X}$, wird eine neue zufällige Trajektorie $\widetilde{Y} = (\widetilde{Y}_0,\dots,\widetilde{Y}_k)$, startend in $\widetilde{Y}_0 = \widehat{1}$, produziert nach den folgenden Übergangswahrscheinlichkeiten zum Zeitpunkt $s \in \{1,\dots,k\}$. Mit einer Zufallsvariable $\widehat{U}_s$, die die Verteilung einer auf $[0,1]$ gleichförmig verteilten Zufallsvariable U gegeben das Ereignis $\{\widetilde{\phi}(\widetilde{X}_{s-1},U) = \widetilde{X}_s\}$ besitzt, setzen wir $\widetilde{Y}_s = \widetilde{\phi}(\widetilde{Y}_{s-1},\widehat{U}_s)$. Falls $\widetilde{Y}_k = \widehat{0}$, so terminiert der Algorithmus und gibt den Wert z aus. Anderen Falls beseitigen wir sämtliche Information und wiederholen die Routine mit dem nächsten Wert von k.

Aus der Struktur des Algorithmus ist sofort ersichtlich, dass er jederzeit abgebrochen und neu gestartet werden darf, und diese Eigenschaft war die hauptsächliche Motivation. Die Werte von k muss so getroffen werden, dass sie divergieren, damit die Akzeptanzwahrscheinlichkeit gegen Eins konvergiert.

Die Monotonie von $\widetilde{\phi}(\,\cdot, u)$ für jedes u und die Konstruktion von $\widetilde{Y}$ garantiert übrigens, dass $\widetilde{X}_s \leq \widetilde{Y}_s$ für jedes $s \in \{0, \dots, k\}$ gilt. Dies sieht man iterativ ein: Es gilt $\widetilde{X}_0 = \widehat{0} \leq \widehat{1} = \leq \widetilde{Y}_0$, und wenn $\widetilde{X}_{s-1} \leq \widetilde{Y}_{s-1}$, so folgt $\widetilde{X}_s = \phi(\widetilde{X}_{s-1}, U_s) \leq \phi(\widetilde{Y}_{s-1}, U_s) = \phi(\widetilde{Y}_{s-1}, \widehat{U}_s) = \widetilde{Y}_s$.

Als Vorstufe zum Korrektheitsbeweis des Fill-Algorithmus werfen wir einen Blick auf das sogenannte *rejection-sampling*:

Lemma 6.5.1 (Verwerfungsmethode). *Sei ν ein Wahrscheinlichkeitsmaß auf I, und sei $c > 1$ so, dass $\pi(i) \leq c\nu(i)$ für jedes $i \in I$. Dann terminiert der folgende Algorithmus in endlicher Zeit und generiert eine nach π verteilte Stichprobe:*

(a) Ziehe einen nach ν verteilten Wert $X \in I$.

(b) Gegeben $X = i$, akzeptiere diesen Wert mit Wahrscheinlichkeit $\frac{\pi(i)}{c\nu(i)}$ und terminiere; ansonsten verwerfe ihn und gehe zurück zu (a).

Beweis. Es gilt

$$\mathbb{P}(X = i, X \text{ wird akzeptiert}) = \nu(i)\frac{\pi(i)}{c\nu(i)} = \frac{\pi(i)}{c},$$

also auch

$$\mathbb{P}(X \text{ wird akzeptiert}) = \frac{1}{c} \qquad \text{und} \qquad \mathbb{P}(X = i \mid X \text{ wird akzeptiert}) = \pi(i).$$

Da die Akzeptanz von X positive Wahrscheinlichkeit hat, tritt sie in endlicher Zeit ein, und der Zeitpunkt des ersten Akzeptierens hat sogar endliche Erwartung. □

Mit diesem allgemeinen Beispiel im Hinterkopf beweisen wir nun die Korrektheit des Fill-Algorithmus.

Satz 6.5.2. *Der Algorithmus von Fill terminiert in endlicher Zeit und gibt einen nach π verteilten Wert aus.*

Beweis. Die Verteilung des vorgeschlagenen Wertes z ist $\nu_k = \mathbb{P}_{\widehat{0}}(X_k \in \cdot) = P^k(\widehat{0}, \cdot)$, wobei P^k die k-te Potenz von P bezeichnet. Wir setzen im Folgenden voraus, dass k so groß ist, dass $\widetilde{P}^k$ ausschließlich positive Einträge besitzt. Aus (6.5.1) erhält man leicht die Beziehung

$$\frac{\pi(i)}{P^k(\widehat{0}, i)} = \frac{\pi(\widehat{0})}{\widetilde{P}^k(i, \widehat{0})}, \qquad i \in I.$$

Wir betrachten nun $c_k = \pi(\widehat{0})/\widetilde{P}^k(\widehat{1}, \widehat{0})$. Auf Grund der Monotonie von $\widetilde{P}$ gilt

$$\pi(i) = \frac{\pi(\widehat{0})}{\widetilde{P}^k(i, \widehat{0})} P^k(\widehat{0}, i) \leq \frac{\pi(\widehat{0})}{\widetilde{P}^k(\widehat{1}, \widehat{0})} P^k(\widehat{0}, i) = c_k \nu_k(i), \qquad i \in I,$$

wir befinden uns also in der Situation von Lemma 6.5.1. Um dieses Lemma anzuwenden, müssen wir zeigen, dass die Akzeptanzwahrscheinlichkeit für den vorgeschlagenen Wert z gleich $\frac{\pi(z)}{c_k \nu_k(z)}$ ist. Hierzu reicht es zu zeigen, dass

$$\mathbb{P}(\widetilde{Y}_k = \widehat{0} \mid X_k = z) = \frac{\widetilde{P}^k(\widehat{1}, \widehat{0})}{\widetilde{P}^k(z, \widehat{0})}, \tag{6.5.2}$$

denn man errechnet leicht, dass

$$\frac{\widetilde{P}^k(\widehat{1},\widehat{0})}{\widetilde{P}^k(z,\widehat{0})} = \frac{\pi(z)\widetilde{P}^k(\widehat{1},\widehat{0})}{\pi(\widehat{0})P^k(\widehat{0},z)} = \frac{\pi(z)}{c_k\nu_k(z)}.$$

Um (6.5.2) einzusehen, analysieren wir die Struktur der gemeinsamen Verteilung der Ketten $\widetilde{Y}$ und $\widetilde{X}$. Für alle $x = (x_0, \dots, x_k) \in I^{k+1}$ mit $x_0 = z$ und $x_k = \widehat{0}$ und für alle $y = (y_0, \dots, y_k)$ mit $y_0 = \widehat{1}$ und $y_k = \widehat{0}$ ist nämlich

$$\begin{aligned}
\mathbb{P}(\widetilde{Y} = y, \widetilde{X} = x) &= \prod_{s=1}^k \widetilde{P}(x_{s-1}, x_s) \prod_{s=1}^k \mathbb{P}(\widetilde{Y}_s = y_s \mid \widetilde{Y}_{s-1} = y_{s-1}, \widetilde{X}_s = x_s, \widetilde{X}_{s-1} = x_{s-1}) \\
&= \prod_{s=1}^k \Big[\mathbb{P}(\widetilde{\phi}(x_{s-1}, U_s) = x_s)\mathbb{P}(\widetilde{\phi}(y_{s-1}, U) = y_s \mid \widetilde{\phi}(x_{s-1}, U) = x_s)\Big] \\
&= \prod_{s=1}^k \mathbb{P}(\widetilde{\phi}(y_{s-1}, U) = y_s) \\
&= \mathbb{P}(\widetilde{Y} = y).
\end{aligned}$$

Durch Summation über alle solchen Vektoren x und y erhält man $\mathbb{P}(\widetilde{Y}_k = \widehat{0}, X_k = z) = \mathbb{P}(\widetilde{Y}_k = \widehat{0})$, und dies impliziert (6.5.2). Dies zeigt, das der Wert z die gesuchte Verteilung π besitzt, wenn er akzeptiert wird.

Da die Akzeptanzwahrscheinlichkeit $\frac{1}{c_k} = \widetilde{P}^k(\widehat{1},\widehat{0})/\pi(\widehat{0})$ für $k \to \infty$ gegen 1 konvergiert, ist sie für genügend große k oberhalb einer festen positiven Schranke, etwa $c_k \geq \frac{1}{2}$. Daher terminiert der Fillsche Algorithmus fast sicher in endlicher Zeit. □

Bibliographie

[AF03] D. Aldous und J.A. Fill, *Reversible Markov Chains and Random Walks on Graphs*, Buch in Vorbereitung, `http://stat-www.berkeley.edu/users/aldous/book.html`.

[B79] L. Babai, Monte-Carlo algorithms in graph isomorphism testing, *Technical Report DMS 79-10*, Département de Mathématique et de Statistique, Université de Montréal, 1979.

[BS85] A. Berretti und A.D. Sokal , New Monte Carlo method for the self-avoiding walk, *J. Stat.Phys.* **40**, 483-531 (1985).

[F98] J.A. Fill, An interruptible algorithm for perfect sampling via Markov chains, *Ann. Appl. Probab.* **8**, 131-162 (1998).

[FR75] R.W. Floyd und R.L. Rivest, Expected time bounds for selection, *Communications of the ACM* **18**, 165-172 (1975).

[F77] R. Freivalds, Probabilistic machines can use less running time, *Information Processing* **77**, Proceedings of IFIP Congress **77**, 839-842 (1977).

[G00] J. Geiger, Algorithmen und Zufälligkeit, Vorlesungsskript, `http://www.mathematik.uni-kl.de/~wwwstoch/jgeiger/`.

[GG84] D. Geman und S. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**, 721-741 (1984).

[G99] O. Goldreich, *Pseudorandomness*, Notices of the American Mathematical Society **46**, 1209-1216 (1999).

[H02] O. Häggström, *Finite Markov Chains and Algorithmic Applications*, Cambridge University Press, 2002.

[Ha70] W.K. Hastings, Monte Carlo sampling methods using Markov chains and their applications, *Biometrika* **57**, 97-109 (1970).

[H61a] C.A.R. Hoare, Algorithm 63 (Partition) and algorithm 65 (Find), *Communications of the ACM* **4**, 321-322 (1961).

[H61b] C.A.R. Hoare, Quicksort, *Computer Journal* **5**, 10-15 (1961).

[Je95] M. Jerrum, A very simple algorithm for estimating the number of k-colorings of a low-degree graph, *Random Structures and Algorithms* **7**, 157-165 (1995).

[KR87] R.M. Karp und M. Rabin, Efficient randomized pattern-matching algorithms, *IBM J. Res. Develop.* **31**, 249-260 (1987).

[K98] D. KNUTH, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3nd edition, Addison-Wesley, Reading, 1998.

[LSZ93] M. LUBY, A. SINCLAIR und D. ZUCKERMAN, Optimal speedup of Las Vegas algorithms, *Inf. Proc. Lett.* **47**, 173-180 (1993).

[MRRTT53] N. METROPOLIS, A.W. ROSENBLUTH, M.N. ROSENBLUTH, A.H. TELLER und E. TELLER, Equation of state calculations by fast computing machines, *Journal of Chemical Physics* **21**, 1087-1092, (1953).

[MR95] R. MOTWANI und P. RAGHAVAN, *Randomized Algorithms*, Cambridge University Press, 1995.

[No97] J. NORRIS, *Markov Chains*, Cambridge University Press, 1997.

[PW96] J.G. PROPP und D.B. WILSON, Exact sampling with coupled Markov chains and applications to statistical mechanics, *Random Structures and Algorithms* **10**, 223-252 (1996).

[R80] M.O. RABIN, Probabilistic algorithms for testing primality, *J. Num. Theor.* **12**, 128-138 (1980).

[RSA78] R.L. RIVEST, A. SHAMIR und L.M. ADLEMAN A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, **21**, 120-126 (1978).

[R91] U. RÖSLER, A limit theorem for 'Quicksort', *Theor. Inf. Appl.* **25**, 85-100 (1991).

[SS77] R. SOLOVAY und V. STRASSEN, A fast Monte Carlo test for primality, *SIAM J. Comput.* **6**, 84-85 (1977).

[Wi00] D. WILSON, How to couple from the past using a read-once source of randomness, *Random Structures and Algorithms* **16**, 85-113 (2000).

[Wi03] D. WILSON, Perfectly random sampling with Markov chains, `http://research.microsoft.com/ dbwilson/exact/`