Zufällige Erzeugung von Catalan-Strukturen

Diplomarbeit von Dorothea Rochusch

betreut von Dr. Stefan Felsner

Fachbereich Mathematik und Informatik
Freie Universität Berlin

8. März 2000

Inhaltsverzeichnis

1 Einleitung

An vielen Stellen wird immer wieder festgestellt, daß deterministische Algorithmen kompliziert sind und lange brauchen, um zu einem Ergebnis zu kommen, während randomisierte Algorithmen oft mit einem sehr einfachen Programm in kurzer Zeit hinreichend gute Ergebnisse liefern. Ein Beispiel dafür ist der oft betrachtete Primzahltest. Durch Randomisierung wird hierbei eine wesentliche Verbesserung der Laufzeit erreicht, während das Ergebnis immer noch mit einer sehr hohen Wahrscheinlichkeit richtig ist. Aus diesem Grund ist man daran interessiert, auch kompliziertere Strukturen, wie z.B. binäre Bäume, zufällig erzeugen zu können. binäre Bäume werden oft eingesetzt, wenn Daten geordnet gespeichert, neu eingefügt oder gelöscht werden sollen. Tritt dabei ein entarteter Baum auf, also einer mit (fast) linearer Höhe, geht die angestrebte Effizienz verloren. Solche entarteten Bäume treten manchmal bei ungünstigen Eingabefolgen auf. Durch Randomisierung gibt es keine ungünstigen Eingabefolgen mehr, so daß das Laufzeitverhalten im Durchschnitt über alle Eingabefolgen gleich ist, was die Effizienz der Datenstruktur verbessert.

Ein weiterer Grund für die Betrachtung von Zufallsgeneratoren für komplexere Strukturen ist die Möglichkeit, die durchschnittlichen Eigenschaften der betrachteten Objekte zu bestimmen. Zum Beispiel haben Triangulierungen eines konvexen n-gons im Durchschnitt einen maximalen Grad (Anzahl von Diagonalen in einer Ecke) von $\log n$. Solche Ergebnisse sind wieder wichtig, um das Laufzeitverhalten anderer Algorithmen zu bestimmen, die mit den erzeugten Strukturen arbeiten.

In dieser Arbeit werden verschiedene Algorithmen zur zufälligen Erzeugung von Catalan-Objekten, wie z.B. den binären Bäumen oder Triangulierungen, vorgestellt und analysiert. Sicherlich werden nicht alle gleichermaßen in der Praxis eingesetzt werden, dennoch liefern sie einen guten Überblick über die Möglichkeiten, Algorithmen zur zufälligen Erzeugung festgelegter Strukturen aufzubauen, wenn nur von einem Zufallsgenerator für ganzzahlige Werte bzw. rationale Zahlen in einem festen Bereich ausgegangen werden kann:

1. Zufällige Erzeugung einer einfacheren Datenstruktur und Umwandlung dieser, so daß die Gleichverteilung erhalten bleibt. Diese Technik verwenden Atkinson und Sack zur Erzeugung von Klammersequenzen (siehe 3.1).

2. Schrittweise Erzeugung der Datenstruktur, so daß insgesamt eine Gleichverteilung erreicht wird. Ein Beispiel sind die schrittweise wachsenden binären Bäume in Abschnitt 4.3 und der Algorithmus von Arnold und Sleep zur Erzeugung von Klammersequenzen (siehe 3.2).

3. Aufstellen einer Rangfunktion und Berechnen der Datenstruktur zu einer

zufällig gewählten Zahl. Verschiedene Rangfunktionen für binäre Bäume finden sich in den Abschnitten 4.1 und 4.2.

4. Erzeugung mit Hilfe einer Markov-Kette wie dem Triangulation-Walk aus Abschnitt 7.1 und einer ähnlichen Flip-Kette für die Dyck-Pfade (siehe 6.1).

Vor allem die Analysetechniken für Markov-Ketten sind allgemeinerer Natur und lassen sich nicht nur bei diesem Problem anwenden. Vor allem bieten sie den Vorteil, daß mit dieser Technik die Strukturen auch relativ leicht mit einer anderen Verteilung als der meistens betrachteten Gleichverteilung erzeugt werden können. In Abschnitt 6.2 wird dies für die Dyck-Pfade gezeigt.

In einigen Anwendungen werden Objekte mit bestimmten Eigenschaften gebraucht. Ein Generator, der alle möglichen Objekte mit gleicher Wahrscheinlichkeit erzeugt, ist dann nicht sinnvoll. In den Abschnitten 7.2 bis 7.4 werden deshalb auch Algorithmen vorgestellt, die Triangulierungen mit ausgewählten Eigenschaften liefern. In der Praxis werden diese Eigenschaften meist nicht bei Triangulierungen gebraucht, sondern die entsprechenden bei Bäumen. Hier zeigt sich, daß es oft nötig oder sinnvoll ist, die Datenstruktur zu wechseln, um einen einfachen Erzeugungsalgorithmus zu finden, und erst dann das generierte Objekt in die eigentlich geforderte Struktur umzuwandeln.

Am Ende der Arbeit finden sich die Programme zu den verschiedenen Algorithmen und eine kurze Analyse zu ihren Einsatzmöglichkeiten in der Praxis.

2 Catalan-Objekte

2.1 Definition der behandelten Catalan-Objekte

Catalan-Objekte sind diejenigen kombinatorischen Strukturen, deren Anzahl-Sequenzen durch die Catalan-Zahlen $C_n = \frac{1}{n+1}\binom{2n}{n}$ gegeben sind. Das sind z.B. die hier betrachteten binären Bäume der Größe n, Triangulierungen eines konvexen $(n+2)$-gons, die gültigen Klammersequenzen aus n Klammerpaaren, sog. Dyck-Pfade usw. (Eine Aufzählung von 66 Catalan-Strukturen findet sich in [23].)

Da diese vier genannten Strukturen in dieser Arbeit behandelt werden sollen, folgen zunächst die genauen Definitionen. Zu den Definitionen sind jeweils die möglichen Strukturen für $n = 3$ angegeben.

Definition 2.1 *Ein **Dyck-Pfad** ist ein Gitterpfad von* $(0,0)$ *nach* $(2n,0)$ *mit den Schritten* $(1,1)$ *(Aufwärtsschritt) und* $(1,-1)$ *(Abwärtsschritt), der nicht unter die x-Achse fällt. Die Menge der möglichen Pfade ist D_n.*

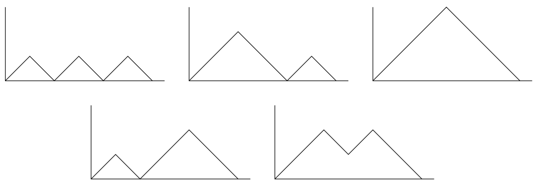

Abbildung 1: Mögliche Dyck-Pfade von $(0,0)$ nach $(2 \cdot 3, 0)$

Definition 2.2 *Eine **gültige Klammersequenz** ist eine Folge aus n "(" und n ")", in der jedes Präfix mindestens so viele "(" wie ")" enthält. Zur leichteren Lesbarkeit wird statt "(" ein λ und statt ")" ein ρ geschrieben. Die Menge der gültigen Klammersequenzen ist K_n.*

((()))   (()())   (())()

()(())   ()()()

Abbildung 2: Mögliche Klammersequenzen mit 3 Klammerpaaren

Definition 2.3 *Sei K ein konvexes Polygon mit $n+2$ Ecken. Eine **Triangulierung** von K ist eine Einteilung von K in n Dreiecke durch Diagonalen von K, die sich nicht schneiden. Die Menge der möglichen Triangulierungen von K ist T_n.*

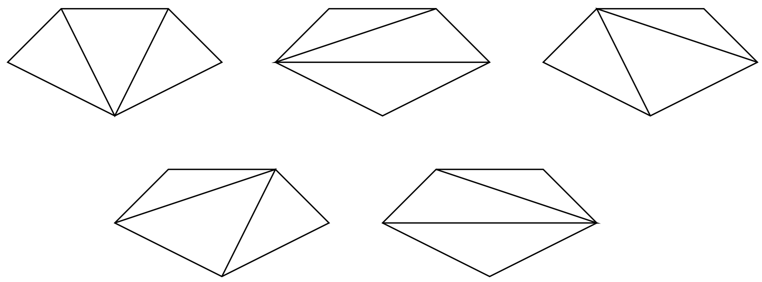

Abbildung 3: Mögliche Triangulierungen eines konvexen (3+2)-Ecks

Definition 2.4 *Ein **binärer Baum** der Größe n ist ein Wurzelbaum mit n inneren Knoten (solche mit zwei Kindern) und $n+1$ Blättern. Die Menge der möglichen binären Bäume der Größe n ist B_n.*

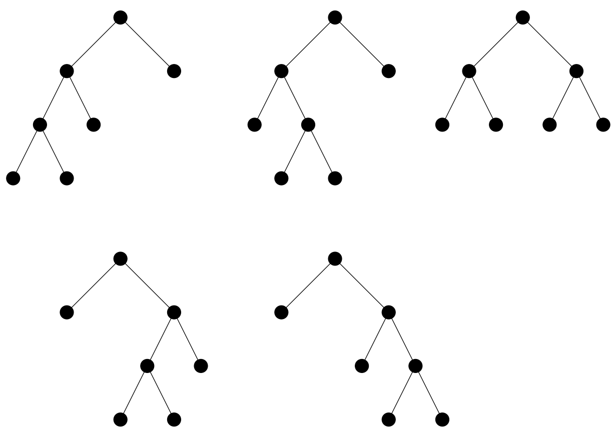

Abbildung 4: Mögliche binäre Bäume mit 3 inneren Knoten

Satz 2.1 *Es gilt:*

$$|T_n| = |B_n| = |K_n| = |D_n| = C_n$$

Beweis: Zuerst wird gezeigt, daß $|D_n| = C_n$ gilt. Die anderen Gleichungen werden dann durch Angabe von Bijektionen zwischen den einzelnen Mengen bewiesen.

1. $|D_n| = C_n$: Sei $1 \leq n < m$. Es gibt $\binom{m+n-1}{n-1}$ Gitterpfade von $(0,1)$ nach (m,n), die nur aus $(1,0)$- und $(0,1)$-Schritten bestehen. Durch Spiegelung des Wegstückes von $(0,1)$ bis zum ersten Schnittpunkt des Weges mit der Winkelhalbierenden (existiert, da $m > n$) an dieser erhält man einen Gitterpfad von $(1,0)$ nach (m,n), der die Winkelhalbierende mindestens einmal schneidet. Aus dieser Bijektion ergibt sich, daß es also auch

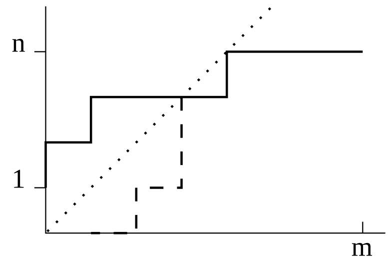

Abbildung 5: Spiegelung an der Winkelhalbierenden

$\binom{m+n-1}{n-1}$ Gitterpfade von $(1,0)$ nach (m,n) gibt, die die Winkelhalbierende mindestens einmal schneiden. Da von $(0,0)$ aus der nächste Punkt entweder $(1,0)$ oder $(0,1)$ ist, folgt daraus: Die Anzahl der Gitterpfade von $(0,0)$ nach (m,n), die die Winkelhalbierende nur in $(0,0)$ schneiden, ist die Anzahl aller Gitterpfade von $(0,0)$ nach (m,n) minus der Anzahl der Gitterpfade von $(0,1)$ nach (m,n) (sie schneiden die Winkelhalbierende immer) minus der Anzahl der Gitterpfade von $(1,0)$ nach (m,n), die die Winkelhalbierende schneiden, also

$$= \binom{m+n}{n} - 2\binom{m+n-1}{n-1} = \frac{m-n}{m+n}\binom{m+n}{n}.$$

Die Anzahl der Gitterpfade von $(0,0)$ nach (n,n), die nie über die Gerade $y = x$ kommen, ist genauso groß wie die Anzahl der Pfade von $(1,0)$ nach $(n+1,n)$, die nie die Winkelhalbierende schneiden, wie in Abbildung 6 zu sehen ist.

Da jeder Pfad von $(0,0)$ aus, der die Winkelhalbierende nicht schneiden soll, mit dem Schritt nach $(1,0)$ beginnt, ist diese Anzahl also gleich der der Gitterpfade von $(0,0)$ nach $(n+1,n)$, die die Winkelhalbierende nicht

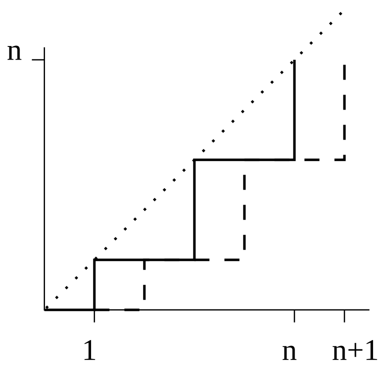

Abbildung 6: Verschiebung eines Pfades

schneiden. Diese ist nach der obigen Gleichung

$$\frac{n+1-n}{n+1+n}\binom{n+1+n}{n} = \frac{1}{n+1}\binom{2n}{n} = C_n.$$

Durch Spiegeln des Gitterpfades an der Winkelhalbierenden und Drehung um 45 Grad erhält man aus einem solchen Pfad von $(0,0)$ nach (n,n), der die Winkelhalbierende nicht überschreitet, einen Dyck-Pfad.

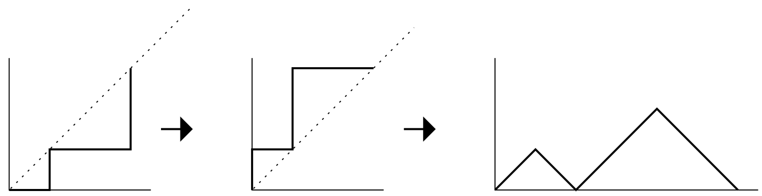

Abbildung 7: Umwandlung in einen Dyck-Pfad

Aus dieser Bijektion folgt damit auch $|D_n| = C_n$.

2. Bijektion $K_n \to D_n$: Für ein Wort k aus K_n mache für jedes λ einen Aufwärtsschritt und für jedes ρ einen Abwärtsschritt, beginnend in $(0,0)$. Da jedes Präfix von k höchstens so viele ρ wie λ enthält, fällt der so entstandene Pfad nie unter die x-Achse. Es werden insgesamt n Aufwärts- und n Abwärtsschritte gemacht, also führt der entstehende Weg von $(0,0)$ nach $(2n,0)$. Zwei verschiedene Klammersequenzen unterscheiden sich an mindestens einer Stelle, also sind die zugehörigen Pfade verschieden. Die Funktion ist also injektiv.

   Genauso entspricht jeder Dyck-Pfad einer gültigen Klammersequenz. Diese Bijektion zeigt: $|K_n| = |D_n| = C_n$.

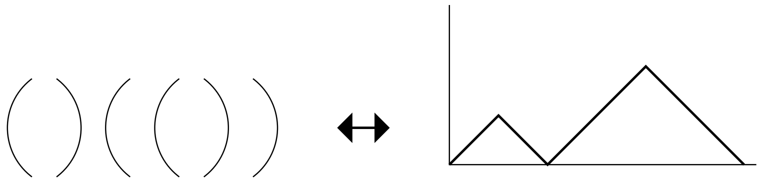

Abbildung 8: Bijektion zwischen Dyck-Pfaden und Klammersequenzen

3. Bijektion $B_n \to K_n$: Für einen Baum b aus B_n kennzeichne jedes linke Kind mit λ und jedes rechte Kind mit ρ. Eine Tiefensuche von b, die jeweils zuerst den linken, dann den rechten Teilbaum besucht, ergibt die entsprechende Klammersequenz. Da jeder Knoten, der ein rechtes Kind hat, auch ein linkes Kind hat, das vorher besucht wurde, ist die entstehende Sequenz gültig. Da die Tiefensuchensequenz einen binären Baum eindeutig bestimmt, ist diese Abbildung injektiv.

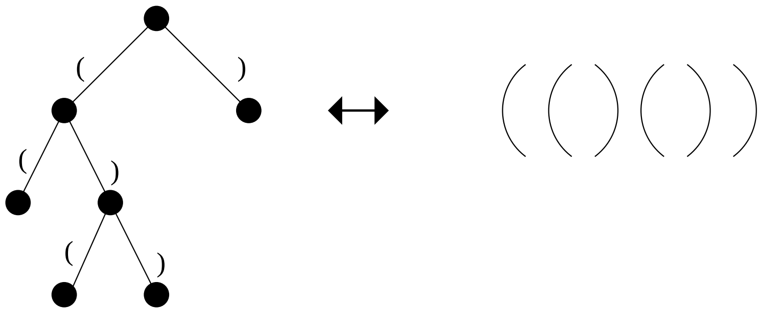

Abbildung 9: Bijektion zwischen Bäumen und Klammersequenzen

   Genauso entspricht jede gültige Klammersequenz einem binären Baum. Diese Bijektion zeigt $|B_n| = |K_n| = C_n$.

4. Bijektion $T_n \to B_n$: Sei K das konvexe $(n+2)$-gon und e_0 eine feste Kante von K. Für eine Triangulierung t von K bildet man den entsprechenden Baum nach folgender Vorschrift: Die Knoten v_e entsprechen den Kanten e der Triangulierung. Die Wurzel ist v_{e_0}. Man entfernt e_0 aus t und erhält 2 oder 0 kantendisjunkte Triangulierungen t_1 und t_2 der beiden Teilpolygone. Die Kanten von t_1 und t_2, die zum Dreieck von e_0 gehörten, werden die Kinder von v_{e_0}. Für die Teilpolygone wird rekursiv weiterverfahren. Der entstehende Baum hat $n-1$ innere Knoten von den Diagonalen von K und einen inneren Knoten, der e_0 entspricht, also insgesamt n. Alle $n+1$ Kanten von K außer e_0 sind Blätter des binären Baumes. Die Eindeutigkeit der Zuordnung läßt sich leicht durch Induktion über die Teilbäume t_1 und t_2 zeigen.

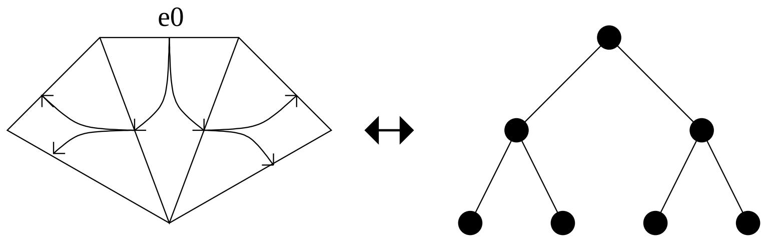

Abbildung 10: Bijektion zwischen Triangulierungen und Bäumen

Umgekehrt entspricht jeder binäre Baum einer Triangulierung. Diese Bijektion zeigt $|T_n| = |B_n| = C_n$.

Bemerkung: Die Bijektion hängt von der Wahl von e_0 ab.

□

Die Programme zur Umwandlung einer Struktur in eine andere mit den angegebenen Bijektionen finden sich in Abschnitt A.1.

2.2 Eigenschaften der Catalanzahlen

Für die Folge der Catalanzahlen wurden viele Eigenschaften bewiesen, benutzt werden in dieser Arbeit nur die zwei folgenden:

Satz 2.2 *Für $n \geq 1$ gilt:*

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$

Beweis: Diese Summenformel läßt sich mit Hilfe der expliziten Formel für C_n nur unter größerem Aufwand nachprüfen. Leichter ist folgende Überlegung: Es gibt C_n binäre Bäume mit n inneren Knoten. Wenn ein solcher Baum i innere Knoten im linken Teilbaum hat, gibt es dafür C_i Möglichkeiten. Für die restlichen $n-1-i$ inneren Knoten im rechten Teilbaum gibt es dann C_{n-1-i} Möglichkeiten, insgesamt also $C_i C_{n-1-i}$. Da ein binärer Baum zwischen 0 und $n-1$ inneren Knoten im linken Teilbaum haben kann, ergibt sich daraus die Summenformel.

□

Die folgende Eigenschaft erlaubt eine rekursive Berechnung der Catalanzahlen, was im Algorithmus `catalanliste` (siehe A) ausgenutzt wird, der eine Liste der ersten n Catalanzahlen liefert:

Satz 2.3 *Es gilt* $C_0 = 1$ *und für* $n \geq 1$*:*

$$C_n = \frac{4n-2}{n+1} \cdot C_{n-1}$$

.

Beweis: $C_0 = \frac{1}{0+1} \begin{pmatrix} 2 \cdot 0 \\ 0 \end{pmatrix} = 1$

$C_n = \frac{1}{n+1} \begin{pmatrix} 2n \\ n \end{pmatrix} = \frac{2n(2n-1)}{2n(2n-1)} \frac{1}{n+1} \frac{(2n)!}{n!n!} = \frac{2(2n-1)}{n+1} \frac{1}{n} \frac{(2n-2)!}{(n-1)!(n-1)!} = \frac{4n-2}{n+1} \cdot C_{n-1}$

$\square$

3 Erzeugung von gültigen Klammersequenzen

3.1 Algorithmus von M.D. Atkinson und J.R. Sack

siehe [4]
Der Algorithmus von M.D. Atkinson und J.R. Sack erzeugt zuerst eine beliebige Klammersequenz aus n λ und n ρ und wandelt diese dann in eine gültige Sequenz um. Durch geeignete Wahl der Transformation wird auch die gewünschte Gleichverteilung gewährleistet. Um dies zeigen zu können, sind einige Definitionen nötig.

Definition 3.1 *Eine Klammersequenz heißt* ***balanciert****, wenn sie gleich viele λ und ρ enthält.*

Definition 3.2 *Eine balancierte Klammersequenz k heißt* ***reduzierbar****, wenn es balancierte, nichtleere Klammersequenzen k_1 und k_2 gibt, so daß $k = k_1k_2$. Sonst heißt k* ***irreduzibel****.*

Definition 3.3 *Eine balancierte Klammersequenz hat* ***Defekt*** *i, wenn der ihr entsprechende Gitterpfad genau $2i$ Schritte unterhalb der x-Achse macht. Der Gitterpfad wird dabei wie in der oben beschriebenen Bijektion berechnet: Beginnend bei $(0, 0)$ bezeichnet λ einen $(1, 1)$-Schritt, ρ einen $(1, -1)$-Schritt.*

Definition 3.4 *Für eine Klammersequenz k bezeichnet k^* die Sequenz, die aus k durch Ersetzung der λ durch ρ und umgekehrt entsteht.*

Beispiel: $k = \lambda\lambda\rho\lambda\rho\rho \to k^* = \rho\rho\lambda\rho\lambda\lambda$

Es gelten folgende Lemmata:

Satz 3.1 *Wenn eine Klammersequenz k balanciert und irreduzibel ist, dann ist k oder k^* gültig.*

Beweis: Durch Induktion über n:

- $n = 1$: Die beiden balancierten, irreduziblen Klammersequenzen der Länge $2n = 2$ sind $k = k^{**} = \lambda\rho$ und $k^* = \rho\lambda$. Die Behauptung gilt.
- $n \geq 2$: Sei k eine balancierte, irreduzible Klammersequenz. Das bedeutet, k läßt sich darstellen als $k = \lambda k_1\rho$ oder $k = \rho k_1\lambda$, wobei k_1 ebenfalls balanciert und irreduzibel ist. $k = \lambda k_1\lambda$ oder $k = \rho k_1\rho$ sind nicht möglich,

da k sonst nicht irreduzibel wäre. Nach Induktionsvoraussetzung ist also k_1 oder k_1^* gültig. Da k irreduzibel ist, muß k_1 im ersten Fall mit λ beginnen, d.h. k_1 ist gültig, im zweiten mit ρ, d.h. k_1^* ist gültig. Damit ist k oder k^* ebenfalls gültig.

□

Satz 3.2 *Eine balancierte Klammersequenz k hat eine eindeutige Zerlegung in irreduzible Sequenzen $k = k_1 k_2 \dots k_r$. Ist k gültig, so sind es auch die k_i, $1 \leq i \leq r$.*

Beweis: Eine Zerlegung von k in irreduzible Teilsequenzen läßt sich leicht nach folgender Regel finden: Beginnend mit Null, wird ein Zähler bei jedem λ in k um 1 herauf, bei jedem ρ um 1 herunter gezählt. Ist man wieder bei Null, ist eine irreduzible Sequenz beendet. Da k selbst balanciert ist, terminiert dieser Algorithmus, wenn der Zähler bei Null ist, also eine Teilsequenz gerade abgeschlossen ist.

Nach Lemma 3.1 ist jeweils die balancierte und irreduzible Teilsequenz k_i oder k_i^* gültig. Wenn k gültig ist, muß jede Teilsequenz mit λ beginnen, woraus folgt, daß die k_i gültig sind.

Angenommen es gibt zwei Zerlegungen von k in irreduzible Sequenzen $k = k_1 k_2 \dots k_{r_k}$ und $k = h_1 h_2 \dots h_{r_h}$. k_i sei die erste Sequenz, die sich von h_i unterscheidet, woraus folgt, daß beide nicht leer sind, weil sonst schon $k_1 k_2 \dots k_{i-1} \neq h_1 h_2 \dots h_{i-1}$ wäre. O.B.d.A. sei k_i die längere von beiden. Dann gibt es eine Sequenz h_i', so daß $k_i = h_i h_i'$. Da k_i und h_i balanciert sind, muß dies auch für h_i' gelten. Also ist k_i nicht irreduzibel im Widerspruch zur Annahme.

□

Definition 3.5 *$K(n)$ ist die Menge aller balancierten Klammersequenzen aus n Klammerpaaren. $K^i(n)$ ist die Menge der balancierten Klammersequenzen mit Defekt i.*

$K(n)$ ist die disjunkte Vereinigung der $K^i(n)$ $(i = 0..n)$. Die gültigen Klammersequenzen sind gerade die mit Defekt Null, also $K^0(n) = K_n$. Durch Angabe einer Funktion $\varphi_n : K(n) \to K_n$, die auf jedem $K^i(n)$ bijektiv ist, wird nun gezeigt, daß alle $K^i(n)$ gleich groß sind. Daraus folgt: Ist eine Zufallsvariable k gleichverteilt in $K(n)$, dann ist $\varphi_n(k)$ gleichverteilt in K_n. Damit ergibt sich dann der folgende Algorithmus:

1. Erzeuge gleichverteilt eine n-elementige Teilmenge M von $\{1 \dots 2n\}$.

2. Erzeuge daraus eine balancierte Klammersequenz $k = k_1 k_2 \dots k_{2n} \in K(n)$ nach der Regel $k_i = \lambda \Leftrightarrow i \in M$.

3. Bilde als Ausgabe die gültige Klammersequenz $\varphi_n(k)$.

Berechnung von φ_n: Die Funktion φ_n wird rekursiv wie folgt definiert:
$\varphi_n : K(n) \to K_n$

$$\varphi_0(\varepsilon) = \varepsilon$$

Für $n > 0$ betrachte zu $k \in K(n)$ die nach Lemma 3.2 eindeutige Zerlegung in balancierte Klammersequenzen $k = xy$, wobei x irreduzibel ist. x besteht aus $2r$ Zeichen, y aus $2s = 2n - 2r$ Zeichen.

$$\varphi_n(k) = \begin{cases} x\varphi_s(y) & \text{, falls } x \text{ gültig ist} \\ \lambda\varphi_s(y)\rho t^* & \text{, falls } x = \rho t \lambda \text{ nicht gültig ist} \end{cases}$$

Mit Induktion folgt, daß $\varphi_n(k)$ wirklich gültig ist, denn im ersten Fall ist x gültig, $\varphi_s(y)$ nach Induktionsvoraussetzung auch, d.h. die Zusammensetzung ist ebenfalls gültig. Im zweiten Fall ist x^* nach Lemma 3.1 gültig und damit auch t^*. Insgesamt ist auch hier die Zusammensetzung gültig.

Behauptung: $\varphi_n|_{K^i(n)}$ ist bijektiv.

Beweis: Angenommen es gibt $k_1, k_2 \in K^i(n)$, die das gleiche Bild unter φ_n haben. Für $j = 1, 2$ sei $k_j = x_j y_j$, wobei die x_j irreduzibel sind. x_j bestehe aus $2r_j$ Zeichen, y_j aus $2s_j$. Dann gibt es drei Möglichkeiten:

1. x_1 und x_2 sind beide gültig. Dann ist

$$x_1\varphi_{s_1}(y_1) = \varphi_n(k_1) = \varphi_n(k_2) = x_2\varphi_{s_2}(y_2).$$

Nach 3.2 gilt dann $x_1 = x_2$ und $\varphi_{s_1}(y_1) = \varphi_{s_2}(y_2)$. Mit Hilfe von Induktion folgt daraus $y_1 = y_2$, also $k_1 = k_2$.

2. x_1 und x_2 sind beide nicht gültig. Sei $x_j = \rho t_j \lambda$. Dann ist

$$\lambda\varphi_{s_1}(y_1)\rho t_1^* = \lambda\varphi_{s_2}(y_2)\rho t_2^*,$$

deren führende Teilsequenzen $\lambda\varphi_{s_j}(y_j)\rho$ irreduzibel sind, also nach 3.2 gleich. Daraus folgt dann auch $t_1^* = t_2^*$, also $t_1 = t_2$ und $x_1 = x_2$. Wieder durch Induktion ist $y_1 = y_2$, insgesamt also $k_1 = k_2$.

3. x_1 ist gültig, x_2 nicht. Sei $x_2 = \rho t_2 \lambda$. Es ist

$$x_1 \varphi_{s_1}(y_1) = \lambda \varphi_{s_2}(y_2) \rho t_2^*,$$

woraus nach 3.2 folgt, daß $x_1 = \lambda \varphi_{s_2}(y_2)\rho$. Das heißt $2r_1 = 2s_2 + 2$ und $2s_1 = 2n - 2r_1 = 2n - (2s_2 + 2) = 2r_2 - 2$. Für den Defekt i der Sequenzen k_j gilt $r_2 \leq i$, da die irreduzible, nicht gültige Klammersequenz x_2 Defekt r_2 hat (bei kleinerem Defekt wäre eine Teilsequenz von x_2 balanciert, also x_2 reduzierbar), und $i \leq s_1$, da x_1 gültig ist. Daraus folgt der Widerspruch

$$i \leq s_1 = r_2 - 1 < r_2 \leq i.$$

Analog für x_2 gültig und x_1 nicht.

Es bleibt zu zeigen, daß φ_n surjektiv auf K_n ist. Klar ist, daß $|K^i(n)| \leq |K_n|$ ist, da φ_n auf $K^i(n)$ injektiv ist. Angenommen eine der Ungleichungen wäre strikt, dann ergibt sich folgender Widerspruch:

$$\binom{2n}{n} = |K(n)| = \sum_{i=0}^{n} |K^i(n)| < (n+1)|K_n| = \binom{2n}{n}.$$

□

Laufzeit: Schritt 1 und 2 des Algorithmus' benötigen eine Laufzeit von $\mathrm{O}(n)$. Zur Berechnung von φ_n wird zuerst ein irreduzibles Präfix der Länge $2r$ bestimmt, wozu $\mathrm{O}(r)$ Zeit erforderlich ist. Dann wird das Bild einer Sequenz der Länge $2n - 2r$ berechnet. Wenn $T(n)$ die Laufzeit für eine Sequenz der Länge $2n$ ist, ergibt sich die Rekursionsgleichung

$$T(n) = \mathrm{O}(r) + T(n-r) = \mathrm{O}(n).$$

Also kann der gesamte Algorithmus in $\mathrm{O}(n)$ ausgeführt werden.

Das Programm zum vorgestellten Algorithmus findet sich in Abschnitt A.2.1.

3.2 Algorithmus von D.B. Arnold und M.R. Sleep

siehe [3]
Während der eben beschriebene Algorithmus sofort eine ganze Klammersequenz erzeugt und sie dann in eine gültige umwandelt, wird die Sequenz in diesem Algorithmus zeichenweise erzeugt, indem jeweils die Wahrscheinlichkeiten berechnet werden, mit der als nächstes Zeichen ein λ bzw. ein ρ steht.
Dafür bezeichne r die Anzahl der λ in der bereits erzeugten Sequenz, für die

noch kein ρ vorhanden ist (also r = Anzahl $\lambda-$ Anzahl ρ), und s die Anzahl der noch zu erzeugenden Zeichen. Man sagt, die erzeugte Teilsequenz befindet sich im Zustand (r, s). Alle möglichen Ergänzungen können durch das Einfügen von r mal ρ in eine gültige Klammersequenz der Länge $s - r$ erzeugt werden. Das heißt, wenn diese beiden Zahlen gegeben sind, sind dadurch auch alle möglichen Ergänzungen der Teilsequenz gegeben, sie hängen nicht von der tatsächlichen Länge der Gesamtsequenz oder vom Aussehen des bereits erzeugten Teils ab. Es sei also $a_{r,s}$ die Anzahl der möglichen Ergänzungen der Länge s einer Teilsequenz mit r zusätzlichen ρ. Von diesen $a_{r,s}$ Ergänzungen beginnen $a_{r+1,s-1}$ mit einem λ und $a_{r-1,s-1}$ mit einem ρ. Das bedeutet, die Wahrscheinlichkeit, daß das nächste Zeichen im Zustand (r, s) ein ρ ist, ist gegeben durch

$$P_{r,s}[\rho] = \frac{a_{r-1,s-1}}{a_{r,s}}.$$

Berechnung der $a_{r,s}$: Zur Berechnung der $a_{r,s}$ wird wieder die Bijektion auf die Gitterpfade benutzt, wobei ein λ einem Aufwärtsschritt, ein ρ einem Abwärtsschritt entspricht. Eine Ergänzung für eine Teilsequenz mit Zustand (r, s) entspricht dann einem Pfad von $A = (0, r)$ zum Punkt $B = (s, 0)$, wobei auch hier jeder Pfad, der nicht unter die x-Achse fällt, zu einer gültigen Ergänzung gehört. Die Anzahl der gültigen Ergänzungen $a_{r,s}$ ist also gleich der Anzahl der Gitterpfa-

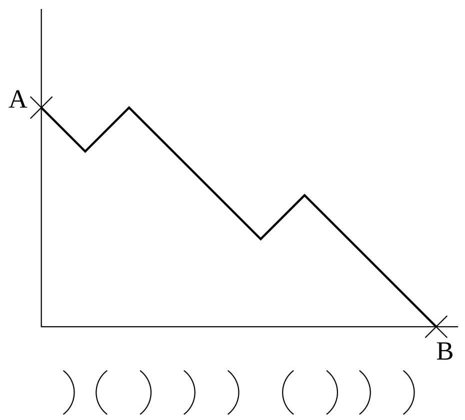

Abbildung 11: Pfad von $A = (0, r)$ nach $B = (s, 0)$

de von A nach B, welche die x-Achse nicht unterschreiten, im folgenden positive Pfade genannt. Die übrigen heißen negativ.
Es sei nun $g_{r,s}$ die Gesamtzahl aller Pfade von A nach B und $n_{r,s}$ die Anzahl der negativen. Dann ist

$$a_{r,s} = g_{r,s} - n_{r,s}.$$

Ein Pfad von A nach B besteht aus p Abwärts- und q Aufwärtsschritten, wobei $s = p + q$ und $r = p - q$ gilt. Daraus ergibt sich $p = \frac{s+r}{2}$ und $q = \frac{s-r}{2}$. Damit ist

$$g_{r,s} = \binom{s}{\frac{s+r}{2}}.$$

Um $n_{r,s}$ zu bestimmen, wird ein Pfad von $A' = (0, -(r+2))$ nach B betrachtet. Jeder dieser Pfade überquert die Linie $y = -1$ mindestens einmal. Sei T der erste dieser Kreuzungspunkte. Durch Spiegelung des Anfangsstückes von A' bis T des Pfades an der Linie $y = -1$ erhält man einen Pfad von A nach B, der negativ ist, da er T enthält. Umgekehrt muß jeder negative Pfad die Gerade $y = -1$ mindestens einmal berühren. So ist eine Bijektion zwischen den negativen Pfaden von A nach B und den Pfaden von A' nach B gegeben. Ein solcher Pfad besteht

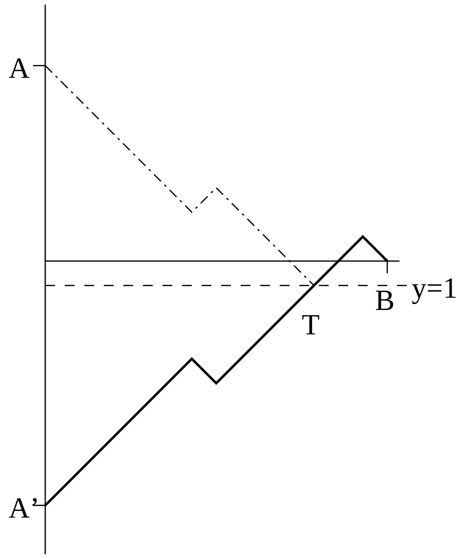

Abbildung 12: Bijektion zwischen negativen Pfaden und Pfaden von A' nach B

aus p Abwärts- und q Aufwärtsschritten mit $s = p + q$ und $r + 2 = q - p$. Daraus erhält man $q = \frac{r+s+2}{2}$ und $p = \frac{s-r-2}{2}$. Also ist $n_{r,s} = \binom{s}{\frac{s+r+2}{2}}$.

Insgesamt ergibt sich dann:

$$a_{r,s} = \binom{s}{\frac{s+r}{2}} - \binom{s}{\frac{s+r+2}{2}} = \frac{2(r+1)}{s+r+2}\binom{s}{\frac{s+r}{2}}$$

Die Wahrscheinlichkeit, daß das nächste Zeichen ein ρ ist, ist dann:

$$P_{r,s}[\rho] = \frac{1}{2}\frac{r(s+r+2)}{s(r+1)}$$

Algorithmus:

1. $s := 2n$ *es sind noch* $2n$ *Zeichen zu erzeugen*
   $r := 0$ *noch kein* λ *ohne* ρ

2. Solange $(s > 0)$:
   Erzeuge mit Wahrscheinlichkeit $P_{r,s}[\rho] = \frac{r(r+s+2)}{2s(r+1)}$ ein ρ bzw. λ
   $s := s - 1$
   Setze $r := r + 1$, falls ein λ erzeugt wurde, sonst $r := r - 1$

Der Algorithmus hat eine Laufzeit, die proportional zur Länge der zu erzeugenden Klammersequenz ist, also $\mathrm{O}(n)$.

Das Programm zum vorgestellten Algorithmus findet sich in Abschnitt A.2.2.

4 Erzeugung von binären Bäumen

4.1 Rangfunktion von G.D. Knott

(siehe [10])

Eine Methode, Strukturen zufällig zu erzeugen, ist, sie durchzunummerieren, eine zufällige Zahl zu generieren und dann das entsprechende Objekt zu berechnen. Dieses Verfahren liegt dem nun behandelten Algorithmus zugrunde.

Definition 4.1 *Für einen binären Baum b bezeichne **anz(b)** die Zahl der inneren Knoten von b, **l(b)** den linken Teilbaum von b und **r(b)** den rechten Teilbaum.*

Um die binären Bäume in B_n durchzunummerieren, wird folgende Ordnungsrelation definiert:
$b_1 < b_2 \Leftrightarrow$
$anz(b_1) < anz(b_2)$
oder $anz(b_1) = anz(b_2)$ und $l(b_1) < l(b_2)$
oder $anz(b_1) = anz(b_2)$ und $l(b_1) = l(b_2)$ und $r(b_1) < r(b_2)$

Diese Ordnung ergibt eine Reihenfolge auf den binären Bäumen aus B_n.

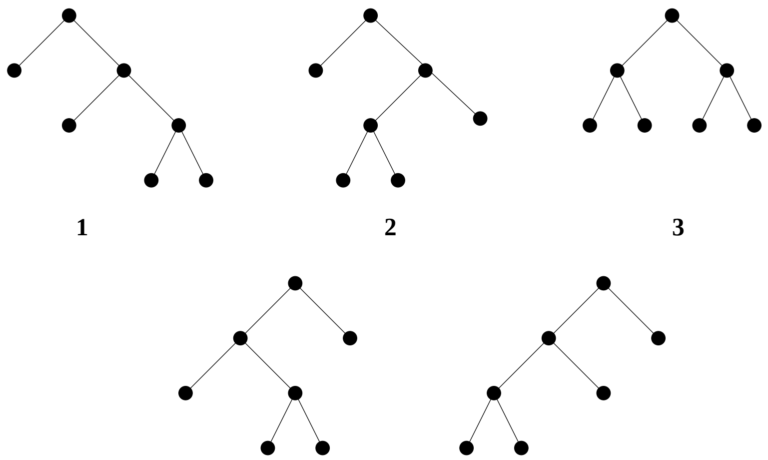

Abbildung 13: Ordnung der Bäume mit $n = 3$ inneren Knoten

Definition 4.2 *$G_{j,n}$ sei die Anzahl binärer Bäume mit n inneren Knoten, von denen j im linken Teilbaum sind.*

Klar sind folgende Eigenschaften von $G_{j,n}$, mit denen sie sich auch leicht berechnen lassen:

1. $G_{j,n} = C_j C_{n-j-1}$
2. $G_{0,0} = 1$ und $G_{j,0} = 0$ für $j > 0$
3. $C_n = \sum_{j=0}^{n-1} G_{j,n}$

Die Nummer eines binären Baumes b ergibt sich aus folgender Rangfunktion:

$$rang(b) = \begin{cases} 1 & \text{, falls } anz(b) = 0 \\ C_{anz(r(b))}(rang(l(b)) - 1) + rang(r(b)) & \\ + \sum_{j<anz(l(b))} G_{j,anz(b)} & \text{, sonst} \end{cases}$$

Diese Funktion kann man auffassen als relative Adresse von $(rang(l(b)), rang(r(b)))$ im 2-dimensionalen Feld der Größe $[1..C_{anz(l(b))}; 1..C_{anz(r(b))}]$ plus der Zahl von Bäumen mit weniger Knoten im linken Teilbaum als b, die gemäß der definierten Ordnung eine kleinere Rangnummer als b haben.

Behauptung: Diese Funktion liefert eine Durchnummerierung der binären Bäume in B_n.

Beweis:

1. $1 \leq rang(b) \leq C_{anz(b)}$: Induktion über $anz(b)$

   $anz(b) = 0 \Rightarrow rang(b) = 1 = C_0$

   Sei $anz(b) > 0$: Dann ist

   $$rang(b) = C_{anz(r(b))}(rang(l(b)) - 1) + rang(r(b)) + \sum_{j<anz(l(b))} G_{j,anz(b)}$$

   Da $rang(l(b))$ und $rang(r(b))$ nach Induktionsvoraussetzung beide mindestens 1 sind, gilt dies auch für $rang(b)$. Außerdem gilt nach Induktionsvoraussetzung $rang(l(b)) \leq C_{anz(l(b))}$ und $rang(r(b)) \leq C_{anz(r(b))}$, also

   $$\begin{aligned} rang(b) &= C_{anz(r(b))}(rang(l(b)) - 1) + rang(r(b)) + \textstyle\sum_{j<anz(l(b))} G_{j,anz(b)} \\ &\leq C_{anz(r(b))}(C_{anz(l(b))} - 1) + C_{anz(r(b))} + \textstyle\sum_{j<anz(l(b))} G_{j,anz(b)} \\ &= C_{anz(r(b))} C_{anz(l(b))} + \textstyle\sum_{j<anz(l(b))} C_j C_{anz(b)-j-1} \\ &= \textstyle\sum_{j\leq anz(l(b))} C_j C_{anz(b)-j-1} \\ &\leq \textstyle\sum_{j=0}^{n-1} C_j C_{anz(b)-j-1} \\ &= C_{anz(b)}, \text{ nach Eigenschaft 2.2 der Catalanzahlen} \end{aligned}$$

2. $rang(b_1) = rang(b_2) \Rightarrow b_1 = b_2$:
Für einen beliebigen Baum $b \in B_n$ gilt

$$rang(b) \in]\sum_{j=0}^{i-1} G_{j,n}, \sum_{j=0}^{i} G_{j,n}] \Leftrightarrow anz(l(b)) = i,$$

denn

$$\begin{array}{rcl} 1 & \leq & C_{anz(r(b))}(rang(l(b)) - 1) + rang(r(b)) \\ & \leq & C_{anz(r(b))}C_{anz(l(b))} \\ & = & G_{anz(l(b)),n} \end{array}.$$

Sei also $rang(b_1) = rang(b_2)$.
Nach der Beobachtung von eben folgt daraus: $anz(l(b_1)) = anz(l(b_2))$ und $anz(r(b_1)) = anz(r(b_2))$
$\Rightarrow \sum_{j<anz(l(b_1))} G_{j,anz(b_1)} = \sum_{j<anz(l(b_2))} G_{j,anz(b_2)} =: x$

$$\begin{array}{rcl} \Rightarrow rang(r(b_1)) & = & (rang(b_1) - x - 1) \mod C_{anz(r(b_1))} + 1 \\ & = & (rang(b_2) - x - 1) \mod C_{anz(r(b_2))} + 1 \\ & = & rang(r(b_2)) \end{array}$$

Mit Induktion über die Zahl der inneren Knoten folgt daraus $r(b_1) = r(b_2)$ und dann auch $l(b_1) = l(b_2)$, also $b_1 = b_2$.

Algorithmus: Um $rang(b)$ zu berechnen, werden die C_j für $0 \leq j \leq n$ gebraucht. Ebenso werden diese Zahlen bei der Umkehrung der Rangfunktion verwendet. Um die Laufzeit des Algorithmus' klein zu halten, sollte also eine Tabelle angelegt werden, die diese Werte enthält. Mit Hilfe der Rekursionsformel $C_n = \frac{4n-2}{n+1} \cdot C_{n-1}$ und $C_0 = 1$ (siehe 2.3) läßt sich die gesamte Tabelle in $\mathrm{O}(n)$ berechnen. Zur Darstellung eines binären Baumes wird hier eine sogenannte Tree-Permutation der Zahlen $1..n$ benutzt. Man betrachtet eine Nummerierung der inneren Knoten des Baumes, so daß alle Nummern im linken Teilbaum eines Knotens kleiner als die ihm zugeordnete Zahl sind, die wiederum kleiner ist als alle Nummern in seinem rechten Teilbaum. Ein Preorder-Durchlauf des Baumes liefert dann eine Permutation p der Zahlen $1..n$ mit der Eigenschaft, daß sie keine Teilsequenz $p_i p_j p_k$ enthält mit $p_k < p_i < p_j$. Diese Permutation beschreibt den Baum eindeutig. Diese Darstellung eines Baumes wird gewählt, da die Nummer eines Knotens k gerade die Anzahl der Knoten in seinem linken Teilbaum plus 1 ist plus eventuell der Nummer des Vaterknotens, wenn k selbst ein rechtes Kind ist. Da der Wert eines Baumes in der Rangfunktion auf den Anzahlen der Knoten in den jeweiligen Teilbäumen beruht, lassen sich diese Zahlen besonders leicht berechnen, so daß sich diese Darstellung anbietet. Der folgende Algorithmus berechnet aus dem Rang eines Baumes b seine Tree-Permutation:

1. $j = \min\{k | \sum_{i=0}^{k} G_{i,n} \geq rang(b)\}$
   j ist die Anzahl der Knoten im linken Teilbaum
   $x = \sum_{i=0}^{j} G_{i,n}$

2. $p_1 := j + 1$ *Nummer der Wurzel*

3. $rang(r(b)) = 1 + (rang(b) - x - 1) \mod C_{n-j-1}$
   $rang(l(b)) = (rang(b) - x - rang(r(b)))/C_{n-j-1} + 1$

4. Berechne die Nummern der j Knoten im linken Teilbaum aus $rang(l(b))$ und die der $n - j - 1$ Knoten im rechten Teilbaum aus $rang(r(b))$, um $p_1 = j + 1$ erhöht.

Der Algorithmus hat eine Laufzeit von $O(n)$, wenn die Erhöhung der Knotennummern in Schritt 4 geschickt durchgeführt wird. Es kann also ein zufälliger binärer Baum ebenfalls in $O(n)$ bestimmt werden.

Das Programm zur Berechnung des Baumes aus seinem Rang nach Knott findet sich in Abschnitt A.3.1.

4.2 Rangfunktion von F. Ruskey und T.C. Hu

siehe [18]

Eine andere Methode, eine Rangfunktion auf den binären Bäumen einzuführen, ist nach der lexikographischen Ordnung der Folgen der Blatt-Level (siehe Abbildung 14). Ein Baum $b \in B_{n-1}$ wird dargestellt durch die Folge der n Level, auf denen sich seine Blätter befinden, von links nach rechts betrachtet. Um eine solche Levelfolge erzeugen zu können, muß zuerst geklärt werden, welche Folgen von n positiven ganzen Zahlen überhaupt die Level-Nummern eines binären Baumes darstellen:

Definition 4.3 *Die Folge $a_1 a_2 \dots a_n$ heißt **gültige Sequenz**, wenn sie den Blattleveln eines Baumes mit n Blättern entspricht. Entsprechend heißt $a_1 a_2 \dots a_j$ **gültige Anfangssequenz** für n, wenn es eine Folge $a_{j+1} \dots a_n$ gibt, die $a_1 a_2 \dots a_j$ zu einer gültigen Sequenz erweitert.*

Definition 4.4 *Eine **Reduktion von links** einer Sequenz ist das Ersetzen des linkesten Paares $a_{k-1} = a_k =: q$ durch $q - 1$, d.h.*

$$a_1 a_2 \dots a_{k-2} q q a_{k+1} \dots a_n \Rightarrow a_1 a_2 \dots a_{k-2} (q-1) a_{k+1} \dots a_n.$$

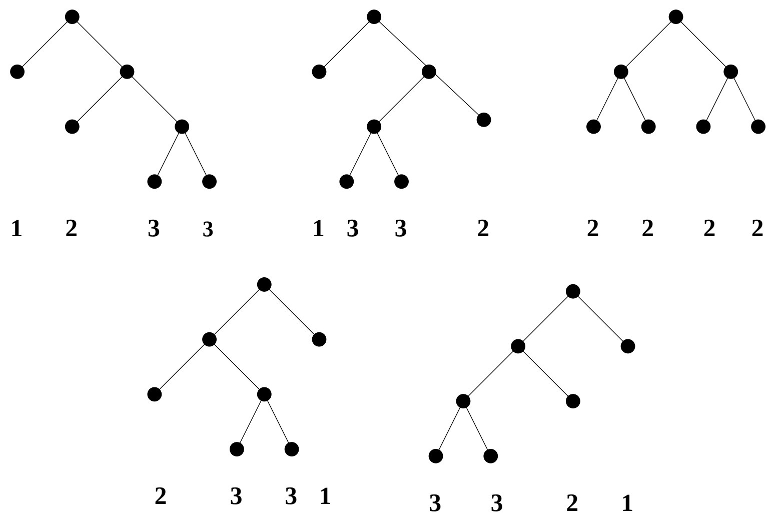

Abbildung 14: Ordnung der Bäume mit $n = 3$ inneren Knoten

Im Baum entspricht eine solche Reduktion der Entfernung der Blätter vom linkesten Knoten, der nur Blätter als Kinder hat. Ist keine Reduktion mehr möglich, spricht man von einer *links-reduzierten Sequenz*. Eine entsprechende Definition gilt für die Reduktion von rechts. Es gelten folgende Lemmata:

Satz 4.1 *Wenn $a_1 a_2 \dots a_n$ gültig ist, gilt:* $\sum_{j=1}^{n} 2^{-a_j} = 1$.

Beweis: Induktion über n:
$n = 1$: Der einzige binäre Baum mit nur einem Blatt besteht nur aus der Wurzel mit Blattlevel 0. Für die einzige gültige Sequenz $a_1 = 0$ gilt also $\sum_{j=1}^{n} 2^{-a_j} = 2^0 = 1$.

$n \geq 2$: Sei b der zu $a_1 a_2 \dots a_n$ gehörige Baum. In b gibt es zwei Blätter v_1 und v_2 mit dem gleichen Vater v, also $a_{v_1} = a_{v_2}$. Ohne diese beiden Blätter entsteht aus b ein Baum mit $n - 1$ Blättern, für dessen Blattlevelsequenz $a_1^* a_2^* \dots a_{n-1}^*$ nach Induktionsvoraussetzung gilt $\sum_{j=1}^{n-1} 2^{-a_j^*} = 1$. Das Level der Blätter v_1 und v_2 ist gerade um 1 größer als das von v, für alle anderen Blätter bleiben die Level gleich. Also ist $2^{-a_{v_1}} + 2^{-a_{v_2}} = 2 \cdot 2^{-(a_v^*+1)} = 2^{a_v^*}$. Damit ist $\sum_{j=1}^{n} 2^{-a_j} = \sum_{j=1}^{n-1} 2^{-a_j^*} = 1$.

□

Satz 4.2 *Eine Sequenz $a_1 a_2 \dots a_n$ ist genau dann gültig, wenn eine Folge von $n - 1$ Reduktionen von links oder rechts zur reduzierten Sequenz 0 führt.*

Beweis: Induktion über n:
$n = 1$: Der einzige binäre Baum mit nur einem Blatt besteht nur aus der Wurzel mit Blattlevel 0. Die zugehörige einzige gültige Sequenz ist $a_1 = 0$.

$n \geq 2$: Sei b der zu $a_1 a_2 \ldots a_n$ gehörige Baum. In b gibt es zwei Blätter v_1 und v_2 mit dem gleichen Vater v. Ohne diese beiden Blätter entsteht aus b ein Baum mit $n-1$ Blättern, dessen Blattlevelsequenz $a_1^* a_2^* \ldots a_{n-1}^*$ sich nach Induktionsvoraussetzung zu 0 reduzieren läßt. Da v_1 und v_2 Nachbarn sind, sind ihre Blattlevel in der Folge $a_1 a_2 \ldots a_n$ ebenfalls Nachbarn, außerdem sind sie gleich. v hat ein Blattlevel, das genau um 1 kleiner ist als das von v_1 bzw. v_2. Also ist $a_1^* a_2^* \ldots a_{n-1}^*$ eine Reduktion der Sequenz $a_1 a_2 \ldots a_n$, die sich deshalb ebenfalls reduzieren läßt. Ist umgekehrt $a_1 a_2 \ldots a_n$ zu Null reduzierbar, dann entspricht die reduzierte Sequenz $a_1^* a_2^* \ldots a_{n-1}^*$ einem Baum, in dem ein Blatt so erweitert werden kann, daß die Levelsequenz des entstehenden Baumes gerade $a_1 a_2 \ldots a_n$ ist.

□

Satz 4.3 *Sei $a_1 a_2 \ldots a_n$ eine Sequenz von positiven ganzen Zahlen, und es gibt ein k mit $a_1 < a_2 < \ldots < a_{k-1}$ und $a_k > a_{k+1} > \ldots > a_n$. Dann gilt:*
$a_1 a_2 \ldots a_n$ ist genau dann gültig, wenn

1. $a_{k-1} = a_k = n - 1$
2. *und die $a_1, a_2, \ldots a_{k-1}, a_{k+1}, \ldots, a_n$ alle verschieden sind.*

Beweis:

1. "$\Leftarrow$" Beweis durch Induktion über n.
   $n = 2$: $a_1 a_2 = 11$ wird reduziert zu 0, ist also gültig.
   $n > 2$: $a_1 \ldots n-1, n-1 \ldots a_n$ kann reduziert werden zu einer Sequenz, die wieder den Bedingungen des Satzes genügt und nur $n-1$ Elemente hat. Nach Induktionsvoraussetzung ist sie also gültig, damit auch $a_1 \ldots n-1, n-1 \ldots a_n$

2. "$\Rightarrow$" $a_1 \ldots a_{k-1} a_k \ldots a_n$ sei gültig. Dann muß es ein zu reduzierendes Paar geben, also $a_{k-1} = a_k \leq n-1$. Nun betrachte man die binäre Darstellung von $\sum_{j=1}^{k-1} 2^{-a_j}$. Es gibt an der j-ten Stelle rechts vom Komma eine 1 genau dann, wenn $a_i = j$ für ein $1 \leq i \leq k-1$. Das gleiche gilt für $\sum_{j=k}^{n} 2^{-a_j}$. Weil diese Summen sich zu 1 addieren (nach Lemma 4.1), kann jeweils nur eine 1 an der gleichen Stelle in beiden Binärdarstellungen vor dem a_k-ten Platz stehen. Daraus folgt, daß Bedingung 2) gilt und $a_k = n-1$, also Bedingung 1).

$\square$

Satz 4.4 *Ist* $a_1 \dots a_n$ *gültig, so auch* $a_1 a_2 \dots a_{n-1} a_n + 1, a_n + 2, \dots a_n + m, a_n + m$ *für jedes* $m > 0$.

Beweis: $a_1 a_2 \dots a_{n-1} a_n + 1, a_n + 2, \dots a_n + m, a_n + m$ kann durch m Reduktionen von rechts auf $a_1 \dots a_n$ reduziert werden.

$\square$

Satz 4.5 $a_1 a_2 \dots a_j$ *ist genau dann gültige Anfangssequenz für* n*, wenn die linksreduzierte Sequenz* $r_1 r_2 \dots r_l$ *folgendes erfüllt:*

1. $1 \leq r_1 < r_2 < \dots r_l$

2. $r_l \leq n - j + l - 1$

Beweis:

1. "$\Leftarrow$" Sei S_i die Sequenz $r_{i+1} - 1, r_{i+1} - 2 \dots r_i + 1$ (mit $r_0 = 0$). S_i ist genau dann leer, wenn $r_{i+1} - r_i = 1$. S_l setze gleich r_l. Nach Bedingung 1) erfüllt $r_1 r_2 \dots r_l S_l S_{l-1} \dots S_1 S_0$ die Voraussetzungen von Lemma 4.3, ist also gültig. Daraus folgt, daß $a_1 \dots a_j$ gültige Anfangssequenz für $r_l + j - l + 1$ ist, denn die Sequenz $S_l S_{l-1} \dots S_0$ besteht aus $r_l - l + 1$ Zahlen, die auch $a_1 a_2 \dots a_j$ erweitern können. Nach Bedingung 2) und Lemma 4.4 auch für n.

2. "$\Rightarrow$" Sei $a_1 a_2 \dots a_j$ gültige Anfangssequenz für n. Falls 1) nicht erfüllt ist, gibt es ein k mit $r_k > r_{k+1}$ und $r_k > r_{k-1}$, falls $k \neq 1$. Da die Levelnummern bei jedem Reduktionsschritt aber nur kleiner werden, kann r_k nie reduziert werden. Also kann $r_1 r_2 \dots r_l a_{j+1} \dots a_n$ nie zu 0 werden.
   2) ist erfüllt, da kein Baum $r_1 r_2 \dots r_l a_{j+1} \dots a_n$ mit $n - j + l$ Blättern ein Blatt auf einem Level größer als $n - j + l - 1$ haben kann.

$\square$

Daraus folgt als Korollar:

Satz 4.6 *Ist* $a_1 a_2 \dots a_j$ $(j < n - 1)$ *gültige Anfangssequenz für* n *mit linksreduzierter Sequenz* $r_1 \dots r_l$*, dann ist* $a_1 a_2 \dots a_j a_{j+1}$ *gültige Anfangssequenz für* n *genau dann, wenn gilt:*

1. $r_l = l \Rightarrow l + 1 \leq a_{j+1} \leq n - j + l - 1$

2. $r_l > l \Rightarrow r_l \leq a_{j+1} \leq n - j + l - 1$

Beweis: Falls die obere Grenze nicht gilt, ist Bedingung 2) von Satz 4.5 verletzt. Falls die untere Grenze nicht gilt, ist Bedingung 1) von Satz 4.5 verletzt. Falls a_{j+1} im angegebenen Bereich liegt, gelten beide Bedingungen, nach Satz 4.5 ist $a_1 a_2 \ldots a_j a_{j+1}$ also gültige Anfangssequenz für n.
Bemerkung: Falls $j = n - 1$ ist, muß $a_n = r_l$ gelten.

$\square$

Die Rang-Funktion Um den Rang einer Level-Folge $a_1 a_2 \ldots a_n$ zu bestimmen, bestimme zuerst für jedes $j = 1, 2, \ldots n$ die Zahl A_j der Elemente, deren erste $j-1$ Komponenten $a_1 a_2 \ldots a_{j-1}$ sind und deren j-te Komponente kleiner ist als a_j. Der Rang von $a_1 a_2 \ldots a_n$ berechnet sich dann als $\sum_{j=1}^{n} A_j$. Wie viele binäre Bäume mit n Blättern gibt es also, deren erste j Levelnummern gerade $a_1 a_2 \ldots a_j$ sind? Sei $r_1 r_2 \ldots r_l$ die linksreduzierte Sequenz von $a_1 a_2 \ldots a_j$. Die gesuchte Zahl ist gleich der Anzahl binärer Bäume mit $n - j + l$ Blättern, deren erste l Levelnummern $r_1 r_2 \ldots r_l$ sind. Nach 4.5 gilt $r_1 < r_2 < \ldots < r_l$, die Levelnummern steigen also echt an. Betrachte den Pfad im Baum mit Levelnummern $r_1 r_2 \ldots r_l a_{j+1} \ldots a_n$ von der Wurzel zum Blatt auf Level r_l. Bei jedem der inneren Knoten auf diesem Pfad können zwei Fälle auftreten:

1. der linke Sohn ist auf dem Pfad, d.h. der rechte Sohn ist Wurzel eines Teilbaums

2. der rechte Sohn ist im Pfad, d.h. der linke Sohn ist ein Blatt, da sonst die Levelnummern nicht echt ansteigen würden.

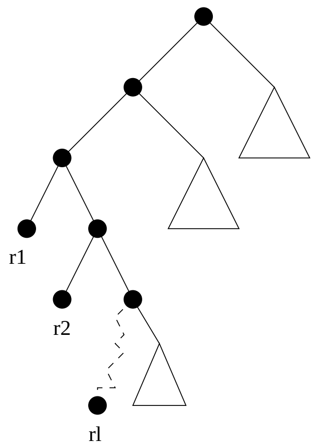

Abbildung 15: Pfad zum Blatt auf Level r_l

Daraus folgt, daß $l-1$ Pfadknoten Blätter als linke Kinder haben und es $r_l - l + 1$ Teilbäume gibt, in denen sich die restlichen $n-j$ Blätter verteilen. Das heißt, die Anzahl der Bäume mit den ersten j Levelnummern $a_1 a_2 \ldots a_j$ ist

$$\sum_{v_1+v_2+\ldots+v_{r_l-l+1}=n-j; v_i \geq 1} C_{v_1-1} C_{v_2-1} \ldots C_{v_{r_l-l+1}-1}.$$

v_i ist dabei die Zahl der Blätter im i-ten Teilbaum. Da es viele Zahlpartitionen von $n-j$ gibt, nämlich $\left(\begin{array}{c} n-j-1 \\ r_l - l \end{array}\right)$, und diese Zahlen deshalb sehr schlecht zu berechnen sind, benutzt man die folgende Konstruktion: Wie man sieht, hängt diese Summe nur von $r_l - l$ und j ab, nicht von der Sequenz $a_1 a_2 \ldots a_j$, also ist folgende Definition möglich:

$$T(n,k) = \sum_{v_1+v_2+\ldots+v_k=n; v_i \geq 1} C_{v_1-1} C_{v_2-1} \ldots C_{v_k-1}$$

ist die Zahl der binären Bäume mit $n+1$ Blättern, deren erste Levelnummer k ist. Wie leicht nachzurechnen ist, gilt:

1. $T(n,n) = 1$
2. $T(n,0) = 0$
3. $T(n,k) = T(n-1,k-1) + T(n,k+1)$

Löst man diese Rekursionsformel, erhält man $T(n,k) = \frac{k}{2n-k} \left(\begin{array}{c} 2n-k \\ n-k \end{array}\right)$. Allerdings ist es leichter, mit den rekursiven Gleichungen, eine Tabelle der $T(n,k)$ anzulegen, die dann für den Algorithmus benutzt werden kann.

Bestimmung der A_j:
$A_n = 0$, da die Sequenz $a_1 a_2 \ldots a_{n-1}$ die letzte Levelnummer a_n nach Satz 4.1 eindeutig festlegt .
Sei also $j < n$: Sei m die kleinste Zahl, so daß $a_1 a_2 \ldots a_{j-1} m$ gültige Anfangssequenz für n ist, d.h. $m = l+1$, falls $r_l = l$ ist, bzw. $m = r_l$ nach 4.6. $r_1 r_2 \ldots r_l$ sei wieder die linksreduzierte Sequenz zu $a_1 a_2 \ldots a_{j-1}$. Nach 4.5 ist A_j die Summe der Anzahlen binärer Bäume mit n Blättern, deren erste j Levelnummern gerade $a_1 a_2 \ldots a_{j-1} m + i$ sind für $i = 0..a_j - m - 1$, also

$$A_j = \sum_{i=0}^{a_j-m-1} T(n-j, m-l+i).$$

Den Rang einer Levelnummern-Folge $a_1 a_2 \ldots a_n$ berechnet auf diese Weise der folgende Algorithmus:

1. $m = 1$ *kleinste Zahl, so daß* $a_1 a_2 \ldots a_{j-1} m = m$ *gültige Anfangssequenz für* n *ist (anfangs ist* $j = 0$*)*
   $l = 0$ *Länge der reduzierten Sequenz*
   $r_0 = 0$

2. $A_j = \sum_{i=0}^{a_j - m - 1} T(n-j, m-l+i)$

3. $l := l + 1$
   $r_l = a_j$
   reduziere $r_1 r_2 \ldots r_l$, solange dies möglich ist

4. falls $r_l = l : m = l + 1$, sonst $m = r_l$ *nach Satz 4.6*

5. $rang(b) = \sum_{j=1}^{n-1} A_j$

Mit der Umkehrung dieses Algorithmus' kann genauso aus dem Rang einer Folge wieder die Sequenz $a_1 a_2 \ldots a_n$ berechnet werden:

1. $r_0 = 0$
   $m = 1$
   $l = 0$

2. für j von 1 bis $n - 1$:
   $i = \min\{k | \sum_{x=0}^{k+1} T(n-j, m-l+x) > rang\}$ $\rightarrow i = a_j - m + 1$
   $rang = rang - \sum_{x=0}^{i} T(n-j, m-l+x)$
   $a_j = m + i - 1$

3. $l = l + 1$
   $r_l = a_j$
   reduziere $r_1 r_2 \ldots r_l$, solange dies möglich ist

4. falls $r_l = l : m = l + 1$, sonst $m = r_l$

5. $a_n = r_l$

Das Programm zur Berechnung eines Baumes aus seinem Rang nach der Blattlevel-Ordnung findet sich in Abschnitt A.3.2.

Bemerkung: Obwohl die Abbildungen 13 und 14 dies nicht nahelegen, unterscheiden sich die beiden Rangfunktionen tatsächlich, dies sieht man jedoch erst bei größeren Beispielen als in den beiden genannten Abbildungen: Nach der lexikographischen Ordnung der Blattlevel ist Baum A größer als Baum B, nach der Ordnung von G.D. Knott stehen sie gerade in umgekehrter Reihenfolge, weil der linke Teilbaum von Baum A weniger Knoten enthält als der von B.

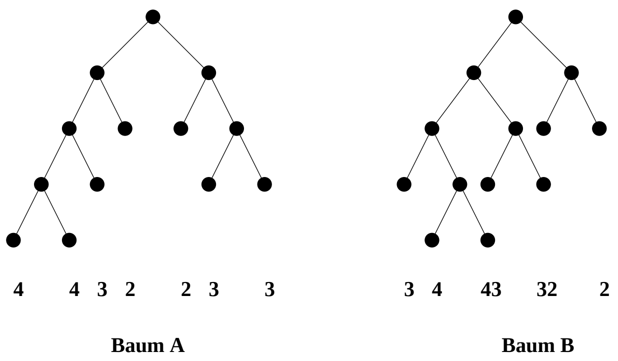

Abbildung 16: Die unterschiedlichen Ordnungen auf den Bäumen

4.3 Schrittweises Wachsen von binären Bäumen

siehe [9]
In diesem Algorithmus wird ein binärer Baum erzeugt, indem schrittweise neue Blätter angehängt werden. Wird der Knoten, der Vater der beiden neuen Blätter werden soll, gleichwahrscheinlich unter allen Blättern ausgewählt, werden leider nicht alle möglichen binären Bäume mit gleicher Wahrscheinlichkeit erzeugt, sondern die mit mittlerer Höhe bevorzugt. Deshalb werden zuerst die Wahrscheinlichkeiten berechnet, nach denen ein Knoten ausgewählt werden muß.

Definition 4.5 *Jeder Baum $b \in B_n$ wird durch die Folge seiner Knoten in Preordnung dargestellt, so daß jeder Baum auch einen eindeutig bestimmten letzten inneren Knoten hat. Die Blätter, die in der Ordnung nach diesem letzten inneren Knoten vorkommen, werden* ***Endblätter*** *genannt, die anderen* ***Nicht-Endblätter****. Die Zahl der Endblätter eines Baumes $b \in B_n$ wird mit* ***e(b)*** *bezeichnet, sie werden gemäß der Preordnung durchnummeriert von 0 bis $e(b) - 1$. Die Menge der Bäume aus B_n mit j Nicht-Endblättern sei B_n^j.*

Erweitert werden soll ein Baum $b_{k-1} \in B_{k-1}$ durch das sogenannte Grafting, d.h. das Anhängen zweier Kinder an eines der Endblätter v mit Nummer x_{k-1}, das dadurch zum inneren Knoten wird. Nach den Regeln der Preordnung ist klar, daß v dann der neue letzte innere Knoten ist. Alle Endblätter nach v bleiben es, die anderen werden zu Nicht-Endblättern, d.h. die Zahl der Endblätter im neuen Baum b_k ist:

$$e(b_k) = e(b_{k-1}) - (x_{k-1} + 1) + 2$$

Die Umkehroperation zum Grafting ist das sogenannte Pruning, also das Entfernen der beiden Blätter vom letzten inneren Knoten. Da jedes Endblatt durch

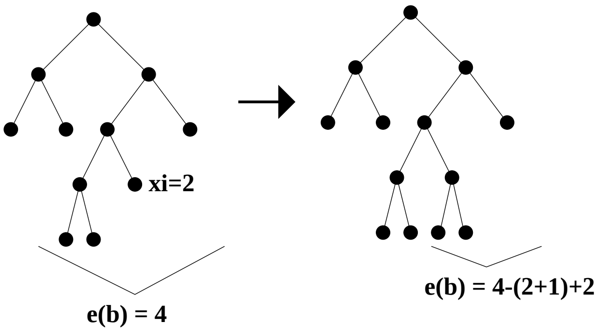

Abbildung 17: Zahl der Endblätter beim Grafting

Grafting erweitert werden kann und somit zum letzten inneren Knoten wird, ist klar:

$$e(b) = |prun^{-1}(b)| \text{ für jeden Baum } b$$

Jeder Baum $b_n \in B_n$ kann nun durch eine Folge $\Delta(b_n) = (x_1, \ldots x_{n-1})$ der Nummern der zu erweiternden Endblätter beschrieben werden, beginnend bei b_1, dem einzigen Baum mit nur einem inneren Knoten. Die Folge hat folgende Eigenschaften:

Satz 4.7 *Für alle $1 \leq i \leq n-1$:*

1. $0 \leq x_i \leq e(b_i) - 1$
2. $e(b_i) = e(b_{i-1}) + 1 - x_{i-1}$
3. $e(b_i) = i + 1 - (x_1 + \ldots + x_{i-1})$
4. $x_1 + \ldots x_{i-1}$ *ist die Zahl der Nicht-Endblätter von* b_i

Die b_i sind dabei die Bäume, die durch die Sequenzen $(x_1, \ldots x_{i-1})$ gegeben sind.

Beweis: 1) und 2) wurden bereits erwähnt. 3) folgt durch Induktion aus 2). Daraus ergibt sich 4), da $i + 1$ die Zahl der Blätter von b_i ist.

$\square$

Aus diesen Eigenschaften folgt, daß die Zahl der Bäume, die durch Grafting aus einem Baum b erzeugt werden können, nur von der Zahl der Endblätter $e(b)$ abhängt und nicht von der Gestalt des Baumes. Es sei also $L(e, j)$ die Zahl der Bäume, die durch j Grafting-Operationen aus einem Baum mit e Endblättern erzeugt werden können. Es gilt für alle $m \geq 2$ und $j \geq 0$:

1. $L(m, 0) = 1$
2. $L(m, 1) = m$
3. $L(2, j) = C_{j+1}$
4. $L(m, j) = L(m-1, j+1) - L(m-2, j+1)$

Beweis: 1) ist klar.
2) begründet sich daraus, daß jedes der m Endblätter erweitert werden kann.
3) ist die Zahl aller Bäume, die durch j Grafting-Operationen aus dem einzigen Baum mit 2 Endblättern, nämlich b_1 hervorgehen, also die Zahl aller Bäume mit $j+1$ inneren Knoten (vergleiche Kapitel 2).
4) Jedes der m Endblätter kann durch die erste Grafting-Operationen erweitert werden. Wird Blatt x_i ausgewählt, ist die Zahl der Endblätter des neuen Baumes nach Satz 4.7 $m + 1 - x_i$, d.h.:

$$\begin{aligned} L(m,j) &= \textstyle\sum_{i=0}^{m-1} L(m+1-i, j-1) \\ &= L(m+1, j-1) + \textstyle\sum_{i=0}^{m-2} L(m-i, j-1) \\ &= L(m+1, j-1) + L(m-1, j) \end{aligned}$$

Durch Indexverschiebung erhält man also $L(m, j) = L(m-1, j+1) - L(m-2, j+1)$.

□

Nach diesen Vorbereitungen können nun die Wahrscheinlichkeiten bestimmt werden, nach denen das zu erweiternde Endblatt ausgewählt wird. Ist b_i der bereits vorhandene Baum mit i inneren Knoten, wird Endblatt Nummer j mit folgender Wahrscheinlichkeit zur Erweiterung ausgewählt:

$$prob(j|b_i) = \frac{L(e(b_i) + 1 - j, n - i - 1)}{L(e(b_i), n - i)}$$

Die Wahrscheinlichkeit, mit der ein bestimmter Baum $b \in B_n$ erzeugt wird, ist also gleich der Wahrscheinlichkeit, daß die ihm entsprechende Folge $(x_1, \ldots x_{n-1})$ ausgewählt wird, also:

$$prob(b) = prob(x_{n-1}|b_{n-1}) prob(x_{n-2}|b_{n-2}) \ldots prob(x_1|b_1) prob(b_1)$$

wobei b_i wieder die Bäume sind, die durch die Sequenz $(x_1, \ldots x_{i-1})$ dargestellt werden. $prob(b_1) = 1$, da immer mit diesem Baum begonnen wird.

$$\begin{aligned} \Rightarrow prob(b) &= \tfrac{L(e(b_{n-1})+1-x_{n-1},0)}{L(e(b_{n-1}),1)} \cdot \tfrac{L(e(b_{n-2})+1-x_{n-2},1)}{L(e(b_{n-2}),2)} \ldots \tfrac{L(e(b_1)+1-x_1,n-2)}{L(e(b_1),n-1)} \cdot 1 \\ &= \tfrac{L(e(b_{n-1})+1-x_{n-1},0)}{L(e(b_1),n-1)}, \text{ da } e(b_{n-i}) + 1 - x_{n-i} = e(b_{n-i+1}) \\ &= \tfrac{1}{L(2,n-1)} \text{ nach Eigenschaft 1 von } L \\ &= \tfrac{1}{C_n} \text{ nach Eigenschaft 3 von } L \end{aligned}$$

Alle binären Bäume aus B_n werden also mit gleicher Wahrscheinlichkeit erzeugt.

Algorithmus:

1. Für i von 1 bis $n-1$:
   $e_i := e_{i-1} + 1 - x_{i-1}$ bzw. $e_1 = 2$ *e_i ist die Zahl der Endblätter in b_i*
   Wähle $x_i \in \{0, \dots e_i - 1\}$ mit Wahrscheinlichkeit $\frac{L(e_i+1-j, n-1-i)}{L(e_i, n-i)}$ für $j \in \{0, \dots e_i - 1\}$

2. Ausgabe : $\Delta(b) = (x_1, \dots x_{n-1})$

Um den Algorithmus auszuführen, sollten die $\frac{n(n-1)}{2}$ Werte für $L(i,j)$ vorher berechnet werden, was mit der Eigenschaft 4 von L leicht möglich ist.

Das Programm zum schrittweisen Wachsen von Bäumen findet sich in Abschnitt A.3.3.

5 Markov-Ketten

Die weiteren Algorithmen arbeiten mit einem völlig anderen Prinzip als die bisher behandelten: Auf der Menge der zu erzeugenden Struktur wird eine Markov-Kette definiert. Der Algorithmus beginnt dann bei einem beliebigen Element der Menge und führt T Schritte der Markov-Kette aus, bis eine annähernde Gleichverteilung auf der Menge erreicht ist. Die Frage, die sich hier stellt, lautet: Wie groß muß T gewählt werden, um eine Gleichverteilung zu gewährleisten? Zur Beantwortung dieser Frage sind einige allgemeinere Definitionen und Sätze notwendig.

5.1 Definitionen und allgemeine Sätze

Definition 5.1 *Eine* ***Markov-Kette*** $\mathcal{M}$ *ist eine Folge von Zufallsvariablen* $(X_t)_{t=0}^{\infty}$ *über einem (in den hier betrachteten Fällen) endlichen Zustandraum* $\Omega \cong \{0 \dots N-1\}$ *mit Transitionsmatrix* $P = (p_{ij})_{i,j=0}^{N-1}$. *Die Matrix* P *ist nicht-negativ und stochastisch, die Übergangswahrscheinlichkeiten sind gegeben durch* $Pr[X_{t+1} = j | X_t = i] = p_{ij}$ *für* $0 \leq i, j \leq N-1$ *und unabhängig von der Zeit* t.

Oft wird solch eine endliche Markov-Kette dargestellt als gerichteter, kanten-gewichteter Graph: Die Knotenmenge ist Ω. Eine Kante (i, j) ist genau dann vorhanden, wenn $p_{ij} > 0$, und wird gewichtet mit p_{ij}.

Definition 5.2 *Die* ***Verteilung*** *von* X_t *wird bezeichnet mit* $\pi^{(t)} = (\pi_i^{(t)})_{i=0}^{N-1}$ *mit* $\pi_i^{(t)} = Pr[X_t = i]$.

$\pi^{(0)}$ bezeichnet die Anfangsverteilung, wobei meist $\pi_i^{(0)} = 1$ für ein $i \in \Omega$, den sogenannten Startwert, und $\pi_j^{(0)} = 0$ für $j \neq i$ gilt. $\pi^{(t)}$ berechnet sich dann durch $\pi^{(t)} = \pi^{(0)} P^t$ für alle $t \in \mathbb{N}$.

Definition 5.3 *Die Markov-Kette* $\mathcal{M}$ *heißt* ***ergodisch***, *wenn es eine Verteilung* $\pi = (\pi_i) > 0$ *gibt, so daß* $\lim_{t \to \infty} (P^t)_{ij} = \pi_j$ *für alle* $i, j \in \Omega$. *In diesem Fall konvergiert* $\pi^{(t)}$ *punktweise gegen* π, *unabhängig von der Anfangsverteilung* $\pi^{(0)}$. *Die Verteilung* π *heißt* ***stationäre Verteilung*** *und erfüllt als einzige die Bedingung* $\pi = \pi P$.

Definition 5.4 *Die Markov-Kette* $\mathcal{M}$ *ist* ***irreduzibel***, *wenn es für jedes Paar von Zuständen* $i, j \in \Omega$ *ein* $t \in \mathbb{N}$ *gibt, so daß* $(P^t)_{ij} > 0$, *d.h.* j *in einer endlichen Zahl von Schritten von* i *aus erreichbar ist. Für den entsprechenden Graphen heißt dies, daß er stark zusammenhängend ist.*

Definition 5.5 *Die Markov-Kette $\mathcal{M}$ heißt **aperiodisch**, wenn $ggt\{t|(P^t)_{ij} > 0\} = 1$ für alle $i, j \in \Omega$.*

Satz 5.1 *Eine endliche Markov-Kette ist genau dann ergodisch, wenn sie aperiodisch und irreduzibel ist.*

Nach Theorem 6.2 in [14].

Definition 5.6 *Eine Markov-Kette $\mathcal{M}$ mit stationärer Verteilung π heißt **reversibel**, wenn $\pi_x P_{xy} = \pi_y P_{yx}$ für alle $x, y \in \Omega$.*

Um abzuschätzen, wie nah die erreichte Verteilung an der gewünschten (Gleich)-Verteilung ist, wird das folgende zeitabhängige Maß eingeführt:

Definition 5.7 *Die **Varianzdistanz** zwischen zwei Verteilungen u und v ist*

$$||u - v|| = \max_{S \subseteq \Omega} |u(S) - v(S)| = \frac{1}{2} \sum_{y \in \Omega} |u(y) - v(y)|$$

.

Definition 5.8 *$\Delta_x(t) = ||P^t_{x,.} - \pi||$ misst die Abweichung der Verteilung zum Zeitpunkt t von der stationären Verteilung.*

Hier interessiert die Zeit, bis die Abweichung von der stationären Verteilung, die in den Beispielen die gewünschte (Gleich-)Verteilung ist, klein genug ist. Dazu wird noch folgende Definition benötigt:

Definition 5.9 *Die **Misch-Zeit** der Markov-Kette $\mathcal{M}$ ist*

$$\tau(\varepsilon) = \max_{x \in \Omega} \tau_x(t)$$

mit $\tau_x(t) = \min\{t|\Delta_x(t') \leq \varepsilon$ für alle $t' \geq t\}$.

Es sind verschiedene Techniken entwickelt worden, um $\tau(\varepsilon)$ abzuschätzen. Benutzt werden z.B. Coupling, wie bei den Dyck-Pfaden, Begrenzung der Conductance des zugrunde liegenden Graphen, hauptsächlich von Sinclair ([20], [21]) beschrieben, und die Comparison-Technik nach [6]. Zur späteren Benutzung dieser Techniken sind noch einige weitere Definitionen und Sätze notwendig.

5.2 Coupling

Definition 5.10 *Ein* ***Coupling*** $(X_t, Y_t) \in \Omega \times \Omega$ *besteht aus zwei Kopien* X *und* Y *der Markov-Kette* $\mathcal{M}$ *mit der Eigenschaft* $X_t = Y_t \Rightarrow X_{t+1} = Y_{t+1}$.
Die ***Couplingzeit*** *ist* $T = \max_{x,y} E[T^{x,y}]$ *mit* $T^{x,y} = \min\{t | X_t = Y_t, X_0 = x, Y_0 = y\}$.

Es gilt der folgende Satz (siehe [1]), der es erlaubt, die Mischzeit mit Hilfe der Couplingzeit abzuschätzen:

Satz 5.2 $\tau(\varepsilon) \leq 8T \log_2 \frac{1}{\varepsilon}$

Beweis: Es sei $d(t) = \max_x \Delta_x(t)$ und $\tau = \tau(\frac{1}{4})$. Weiter sei $\rho_{x,y}(t) = ||P^t_{x,.} - P^t_{y,.}||$ und $\rho(t) = \max_{x,y} \rho_{x,y}(t)$.

Da $P^t_{x,.}$ für jeden Startwert x gegen π konvergiert, gilt $d(t) \leq \rho(t)$, wenn t groß genug ist. Nach der Dreiecksungleichung für die Varianzdistanz gilt außerdem $\rho(t) \leq 2d(t)$.

Um den Satz zu beweisen, wird zuerst folgende Hilfsbehauptung gebraucht:

Behauptung: $\rho(s + t) \leq \rho(s) + \rho(t)$

Hilfslemma:

1. Haben Z_1 und Z_2 Verteilungen v_1 bzw. v_2, dann ist $||v_1 - v_2|| \leq Pr(Z_1 \neq Z_2)$.
2. Sind umgekehrt v_1 und v_2 gegeben, gibt es (Z_1, Z_2) mit $||v_1 - v_2|| = Pr(Z_1 \neq Z_2)$, wobei Z_i Verteilung v_i hat.

ohne Beweis.

Beweis zur Hilfsbehauptung: Seien x, y, s, t fest. Konstruiere $(Z_1(s), Z_2(s))$ so, daß $Z_i(s)$ Verteilung von X_s hat mit $X_0 = x$ bzw. $X_0 = y$. Nach dem Hilfslemma ist diese Konstruktion so möglich, daß $Pr(Z_1(s) \neq Z_2(s)) = \rho_{x,y}(s)$.

Für die Ereignisse $A_j = \{Z_1(s) = j = Z_2(s)\}$ konstruiere $(Z_1(s + t), Z_2(s + t))$ so, daß $Z_1(s + t) = Z_2(s + t)$ und $Pr(Z_1(s + t) = i | A_j) = P^t_{j,i}$. Das heißt, wenn Z_1 und Z_2 zusammentreffen, bewegen sie sich gemeinsam weiter wie die ursprüngliche Markov-Kette, begonnen im Treffpunkt.

Für die Ereignisse $A_{j,k} = \{Z_1(s) = j \text{ und } Z_2(s) = k\}$ $(j \neq k)$ konstruiere $(Z_1(s + t), Z_2(s + t))$ so, daß $Pr(Z_1(s + t) = i | A_{j,k}) = P^t_{j,i}$ und

$Pr(Z_2(s+t) = i|A_{j,k}) = P^t_{k,i}$. Nach dem Hilfslemma ist dann $P(Z_1(s+t) \neq Z_2(s+t)|A_{j,k}) = \rho_{j,k}(t) \leq \rho(t)$.

Da $Z_i(s+t)$ die Verteilung von X_{s+t} hat mit $X_0 = x$ bzw. $X_0 = y$, gilt:

$$\rho_{x,y}(s+t) \leq Pr(Z_1(s+t) \neq Z_2(s+t)) \leq \rho(t)Pr(Z_1(s) \neq Z_2(s)) \leq \rho(t)\rho(s)$$

□

Daraus folgt: $\rho(t) \leq (\rho(t_0))^n$ für $nt_0 \leq t \leq (n+1)t_0$, also $\rho(t) \leq (\rho(t_0))^{\frac{t}{t_0}-1}$.
Für $t_0 = \tau$:

$$\rho(t) \leq \rho(\tau)^{\frac{t}{\tau}-1} \leq (2d(\tau))^{\frac{t}{\tau}-1} \leq (2\frac{1}{4})^{\frac{t}{\tau}-1} = 2^{1-\frac{t}{\tau}}$$

$$\begin{aligned} &\Rightarrow \quad d(t) \leq 2^{1-\frac{t}{\tau}} \\ &\Rightarrow \quad \tau(\varepsilon) \leq \tau(1 + \log_2 \tfrac{1}{\varepsilon}) \end{aligned}$$

(*)

Betrachtet man nun ein Coupling (X_t, Y_t) mit Startwerten x und y, gilt nach dem Hilslemma $\rho_{x,y}(t) \leq Pr(X_t \neq Y_t) = Pr(T^{x,y} > t) \leq \frac{T}{t}$ nach der Markov-Umgleichung. Das bedeutet: $\rho(\tau) \leq \frac{T}{\tau}$, also $\tau \leq \frac{T}{\rho(\tau)} \leq \frac{T}{d(\tau)} = 4T$. Eingesetzt in (*) erhält man damit:

$$\tau(\varepsilon) \leq 4T(1 + \log_2 \frac{1}{\varepsilon}) \leq 8T \log_2 \frac{1}{\varepsilon}$$

□

5.3 Comparison-Technik

Angewendet werden soll im Kapitel über Triangulierungen ein Satz von Randall und Tetali (siehe [16]), der die Mischzeit einer Markov-Kette $\mathcal{M}$ mit Übergangsmatrix P mit Hilfe der bereits bekannten Mischzeit einer zweiten Markov-Kette $\widetilde{\mathcal{M}}$ mit Übergangsmatrix $\widetilde{P}$ abschätzt. Dazu ist es nötig, für jedes Paar (x, y) von Pfaden mit $\widetilde{P}(x, y) > 0$ einen Pfad γ_{xy} zu definieren. Er besteht aus einer Sequenz $x = x_0, x_1, \ldots x_k = y$ mit $P(x_i, x_{i+1}) > 0$. Die Länge eines solchen Pfades ist $|\gamma_{xy}| = k$. Weiter bezeichne $\Gamma(v, w) = \{(x, y)|\widetilde{P}(x, y) > 0, (v, w) \in \gamma_{xy}\}$ die Menge der Pfade, in denen die Kante (v, w) benutzt wird. Mit diesen Festlegungen lautet der Satz:

Satz 5.3 *Für zwei Markov-Ketten $\mathcal{M}$ und $\widetilde{\mathcal{M}}$ auf dem gleichen Zustandsraum Ω mit der gleichen stationären Verteilung π gilt:*

$$\tau(\varepsilon) \leq \frac{4 \log \frac{1}{\varepsilon \pi_*}}{\log \frac{1}{2\varepsilon}} \cdot A(\Gamma) \cdot \widetilde{\tau}(\varepsilon)$$

wobei

$$A(\Gamma) = \max_{P(v,w)>0} \{ \frac{1}{\pi_v P(v,w)} \sum_{\Gamma(v,w)} |\gamma_{xy}| \widetilde{\pi}_x \widetilde{P}(x,y) \}$$

und $\pi_* = \min_{x \in \Omega} \pi_x$.

Beweis: Der Satz folgt aus zwei bekannten Sätzen (siehe [21]):

Satz 5.4 *Für $\varepsilon > 0$ gilt:*

1. *für alle $x \in \Omega$: $\tau_x(\varepsilon) \leq \frac{1}{1-\lambda_1} \log \frac{1}{\pi_x \varepsilon}$*
2. $\max_x \tau_x(\varepsilon) \geq \frac{\lambda_1}{2(1-\lambda_1)} \log \frac{1}{2\varepsilon}$

λ_1 ist dabei der zweitgrößte Eigenwert der Übergangsmatrix P.

Nach Teil 2 von Satz 5.4 gilt für $0 < \varepsilon < 1$:

$$\widetilde{\tau}(\varepsilon) \geq \frac{\lambda_1(\widetilde{P})}{2(1 - \lambda_1(\widetilde{P}))} \log \frac{1}{2\varepsilon}$$

Daraus folgt:

$$(1 - \lambda_1(\widetilde{P})) \geq \frac{1}{4\widetilde{\tau}(\varepsilon)} \log \frac{1}{2\varepsilon}$$

Es wurde benutzt, daß $\lambda_1(\widetilde{P}) \geq \frac{1}{2}$ gilt (sonst können Self-Loops mit Wahrscheinlichkeit $\frac{1}{2}$ eingefügt werden, wie es bei den in Kapitel 7 betrachteten Markov-Ketten schon der Fall ist).

Mit dem Vergleichstheorem (Satz 5.5) erhält man:

$$(1 - \lambda_1(P)) \geq \frac{1}{A(\Gamma)}(1 - \lambda_1(\widetilde{P})) \geq \frac{1}{A(\Gamma)} \frac{1}{4\widetilde{\tau}(\varepsilon)} \log \frac{1}{2\varepsilon}$$

Nach Teil 1 von Satz 5.4 kann die Mischzeit von P, gestartet bei beliebigem $x \in \Omega$, abgeschätzt werden:

$$\tau_x(\varepsilon) \leq \frac{4 \log \frac{1}{\varepsilon \pi_x}}{\log \frac{1}{2\varepsilon}} A(\Gamma) \widetilde{\tau}(\varepsilon)$$

Durch Ersetzen von π_x durch π_* erhält man die Aussage des Satzes.

□

Das Vergleichs-Theorem von Diaconis und Saloff-Coste (siehe [6]) sagt aus:

Satz 5.5

$$(1 - \lambda_1(P)) \geq \frac{1}{A(\Gamma)}(1 - \lambda_1(\widetilde{P}))$$

mit dem oben definierten $A(\Gamma)$.

Beweis: Ω ist eine endliche Menge, und beide Markov-Ketten P und $\widetilde{P}$ sind reversibel. Dann kann man auf der Menge $l^2(\Omega)$ der linearen Abbildungen auf Ω ein Skalarprodukt definieren durch $< f, g >= \sum_{x\in\Omega} f(x)g(x)\pi_x$. Durch die Reversibilität von P ist die Funktion $f \to Pf$ mit $Pf(x) = \sum_{y\in\Omega} f(y)P_{xy}$ selbstadjungiert mit den gleichen Eigenwerten wie P. Diese Eigenwerte können durch die Dirichlet-Form

$$\mathcal{E}(f, f) =< (I - P)f, f >= \frac{1}{2} \sum_{x,y\in\Omega} (f(x) - f(y))^2 \pi_x P_{xy}$$

beschrieben werden:
Sei W ein Unterraum von $l^2(\Omega)$. Dann ist

$$M_{\mathcal{E}}(W) = \max\{\mathcal{E}(f, f) | ||f||_2 = 1, f \in W\}$$

Die übliche Min-Max-Charakterisierung der Eigenwerte (siehe [8]) liefert

$$1 - \lambda_1(P) = \min\{M_{\mathcal{E}}(W)| \dim W = 2\}$$

Definiert man mit Hilfe der zweiten Markov-Kette $\widetilde{P}$ eine zweite Dirichlet-Form $\widetilde{\mathcal{E}}$, so erhält man genauso $1 - \lambda_1(\widetilde{P}) = \min\{M_{\widetilde{\mathcal{E}}}(W)| \dim W = 2\}$. Falls nun gilt $\widetilde{\mathcal{E}} \leq A(\Gamma)\mathcal{E}$, dann folgt daraus das gewünschte $(1 - \lambda_1(P)) \geq \frac{1}{A(\Gamma)}(1 - \lambda_1(\widetilde{P}))$. Zu zeigen ist also nur noch $\widetilde{\mathcal{E}} \leq A(\Gamma)\mathcal{E}$:

$$\begin{aligned}
\widetilde{\mathcal{E}}(f, f) &= \tfrac{1}{2} \textstyle\sum_{x,y\in\Omega} (f(x) - f(y))^2 \widetilde{\pi}_x \widetilde{P}_{xy} \\
&= \tfrac{1}{2} \textstyle\sum_{x,y\in\Omega} (\sum_{(v,w)\in\gamma_{xy}} f(w) - f(v))^2 \widetilde{\pi}_x \widetilde{P}_{xy} \\
&\leq \tfrac{1}{2} \textstyle\sum_{x,y\in\Omega} |\gamma_{xy}| \widetilde{\pi}_x \widetilde{P}_{xy} \sum_{(v,w)\in\gamma_{xy}} |f(w) - f(v)|^2 \\
&\leq \tfrac{1}{2} \textstyle\sum_{P(v,w)>0} |f(w] - f(v)|^2 \frac{\pi_x P(v,w)}{\pi_x P(v,w)} \sum_{\gamma_{xy}\ni(v,w)} |\gamma_{xy}| \widetilde{\pi}_x \widetilde{P}_{xy} \\
&\leq A(\Gamma)\mathcal{E}(f, f)
\end{aligned}$$

□

6 Erzeugung von Dyck-Pfaden

6.1 Gleichverteilung

Der hier vorgestellte Algorithmus von M. Luby, D. Randall und A. Sinclair ([11]) arbeitet mit einer Markov-Kette auf den Dyck-Pfaden. Mit Hilfe von Coupling-Argumenten wird gezeigt, daß $O(n^3 \log n)$ Schritte der Kette ausgeführt werden müssen, bis eine Gleichverteilung erreicht ist. Das ist zwar viel schlechter als die Laufzeiten der oben behandelten Algorithmen, ist aber wegen der Einfachheit des Markov-Ketten-Algorithmus' trotzdem von Interesse.

Die Markov-Kette hat als Zustände die Dyck-Pfade von $(0,0)$ nach $(0,2n)$. Einen Übergang von einem Dyck-Pfad X zu einem anderen Y kann es genau dann geben, wenn sie sich nur durch einen sogenannten **Flip** unterscheiden, d.h. es gibt einen Punkt $0 < p < 2n$, in dem X ein Tal und Y eine Spitze hat oder umgekehrt, während die restlichen Pfade gleich sind.

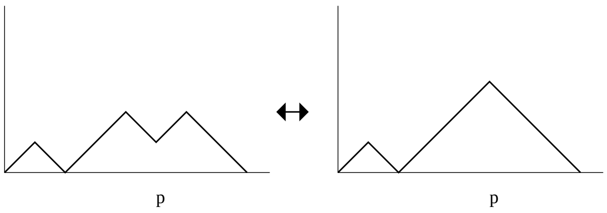

Abbildung 18: Ein Flip

Simuliert wird ein Schritt dieser Kette, indem $p \in \{1 \ldots 2n-1\}$ und $r \in \{\uparrow, \downarrow\}$ gleichverteilt gewählt werden. Falls in p eine Spitze ist und $r = \downarrow$, wird die Spitze durch ein Tal ersetzt, bzw. falls in p ein Tal ist und $r = \uparrow$, wird es durch eine Spitze ersetzt. In allen anderen Fällen bleibt der bisherige Zustand erhalten. Die Übergangsmatrix lautet dann:

$$P_{X,Y} = \begin{cases} \frac{1}{2(2n-1)} & \text{, falls } X \text{ und } Y \text{ sich nur durch Flip unterscheiden} \\ 1 - \sum_{Z \neq X} P_{X,Z} & \text{, falls } X = Y \\ 0 & \text{, sonst} \end{cases}$$

Diese Markov-Kette ist aperiodisch, da die Wahrscheinlichkeit in einem Zustand zu bleiben immer größer Null ist, woraus folgt, daß $\mathrm{ggt}\{t | (P^t)_{ij} > 0\} = 1$ für alle Zustände i, j ist. Sie ist irreduzibel, da jeder Pfad von jedem anderen aus durch Flips erreicht werden kann. Also ist sie ergodisch mit der gewünschten Gleichverteilung als stationärer Verteilung, wie leicht nachzurechnen ist. Ebenso sieht man, daß die Kette reversibel ist.

Man betrachtet nun zwei Kopien der Markov-Kette, deren Zustände die Dyck-Pfade X und Y sind. Jeder enthält $N = 2n - 1$ innere Punkte. Um einen Schritt der Markov-Kette auszuführen, wird nun $(p, r) \in \{1..N\} \times \{\uparrow, \downarrow\}$ gleichverteilt gewählt, und es wird bei beiden Kopien versucht, den entsprechenden Flip auszuführen.
Auf den Dyck-Pfaden kann nun eine partielle Ordnung definiert werden, nämlich $X \succeq Y \Leftrightarrow$ für alle i zwischen 1 und N liegt der i. Punkt von X auf oder über dem i. Punkt von Y. Es ist leicht zu sehen, daß ein Schritt der Markov-Kette die Ordnung auf X und Y erhält. Wegen dieser Beobachtung genügt es, im folgenden nur Pfade zu betrachten, die bezüglich der definierten Ordnung vergleichbar sind, und sich später auf das minimale und maximale Element als Startwerte der Markov-Ketten zu beschränken.

Zur Bestimmung der Mischzeit $\tau(\varepsilon)$ wird ein Coupling $(X_t, Y_t) \in D_n \times D_n$ definiert. Um ein Maß zu haben, wie weit X_t und Y_t noch voneinander entfernt sind, wird als Distanzfunktion $\varphi(X, Y)$ der gewichtete Abstand zwischen den beiden Gitterpfaden X und Y benutzt. Für $x = 0 \ldots 2n$ sei also $h_u(x)$ die Höhe des unteren Pfades an der Stelle x und $h_o(x)$ die Höhe des oberen Pfades. Weiter sei $g(x) = h_o(x) - h_u(x)$ der Abstand der beiden Gitterpfade an der Stelle x. Die eigentliche Distanzfunktion ergibt sich dann durch $\varphi(X, Y) = \sum_{x=0}^{2n}(g(x) \sin \frac{\pi x}{2n})$. Da nur Pfade betrachtet werden, die bezüglich der definierten Ordnung vergleichbar sind, ist $g(x) \geq 0$ für alle interessierenden Pfad-Paare, und für φ gilt: $\varphi(X, Y) \geq 0$ und $\varphi(X, Y) = 0 \Leftrightarrow X = Y$. Zur Verkürzung der Schreibweise sei $\Phi(t) = \varphi(X_t, Y_t)$.

Satz 6.1 *Für zwei Pfade X_t und Y_t mit $X_t \leq Y_t$ gilt:*

$$E[\Delta\Phi | X_t, Y_t] \approx -\frac{1}{2}\frac{1}{2n-1}\left(\frac{\pi}{2n}\right)^2 \Phi(t)$$

.

Beweis: Wenn Punkt p zum Flippen ausgewählt wird, gibt es 4 Möglichkeiten, die in Abbildung 19 zu sehen sind. In den mittleren beiden Fällen wird nicht geflippt, also bleibt die Höhe in p gleich, nämlich $h_o(p) = \frac{h_o(p-1)+h_o(p+1)}{2}$. In den anderen beiden Fällen, wenn $h_o(p) = h_o(p-1) + / - 1$ und $h_o(p+1) = h_o(p-1)$, wird nur geflippt, wenn r entsprechend ausgewählt wird, also mit Wahrscheinlichkeit $\frac{1}{2}$. Dann ändert sich die Höhe in p jeweils um $+2$ oder -2. Auch hier ergibt sich der Erwartungswert $\frac{h_o(p-1)+h_o(p+1)}{2}$, insgesamt also für die neue Höhe des Pfades in Punkt p ebenfalls $E[h_o(p)] = \frac{h_o(p-1)+h_o(p+1)}{2}$, entsprechend für $E[h_u(p)] = \frac{h_u(p-1)+h_u(p+1)}{2}$.

Weiter sei $g'(x)$ der Wert der Funktion g nach einem Flip für $1 \leq x \leq 2n-1$. Mit Wahrscheinlichkeit $\frac{2n-2}{2n-1}$ wird x nicht für den Flip ausgewählt, d.h. $g'(x) = g(x)$.

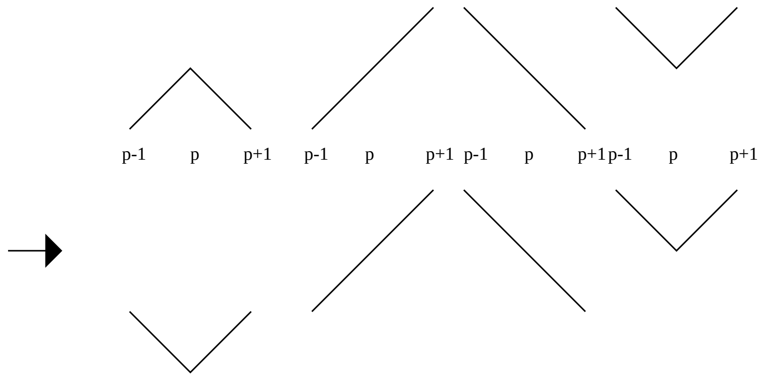

Abbildung 19: Mögliche Situationen in Punkt p

Wird x ausgewählt, ändert sich $g(x)$ in $E[h_o(x) - h_u(x)]$. Insgesamt ist also

$$\begin{aligned} E[g'(x)] &= \tfrac{2n-2}{2n-1} g(x) + \tfrac{1}{2n-1}\left(\tfrac{h_o(x-1)+h_o(x+1)}{2} - \tfrac{h_u(x-1)+h_u(x+1)}{2}\right) \\ &= g(x) - \tfrac{1}{2}\tfrac{1}{2n-1}(2g(x) - g(x+1) - g(x-1)) \end{aligned}$$

Für $x = 0$ und $x = 2n$ ist $g(x)$ immer Null, also auch $E[g'(x)] = 0$.

Nun kann auch die erwartete Änderung von Φ bestimmt werden.

$$\begin{aligned} E[\Phi(t+1)|X_t, Y_t] &= \textstyle\sum_{x=0}^{2n} E[g'(x)] \sin \frac{\pi x}{2n} \\ &= \textstyle\sum_{x=0}^{2n} g(x) \sin \frac{\pi x}{2n} \\ &\quad -\tfrac{1}{2}\tfrac{1}{2n-1} \textstyle\sum_{x=1}^{2n-1} (2g(x) - g(x+1) - g(x-1)) \sin \frac{\pi x}{2n} \\ &= \Phi(t) \\ &\quad -\tfrac{1}{2}\tfrac{1}{2n-1} \textstyle\sum_{x=1}^{2n-1} (2g(x) - g(x+1) - g(x-1)) \sin \frac{\pi x}{2n} \end{aligned}$$

Mit $g(0) = g(2n) = 0$ und $\sin 0 = \sin \pi = 0$ ergibt sich dann:

$$\begin{aligned} E[\Delta\Phi|X_t, Y_t] &= -\tfrac{1}{2}\tfrac{1}{2n-1} \textstyle\sum_{x=1}^{2n-1} (2g(x) - g(x+1) - g(x-1)) \sin \frac{\pi x}{2n} \\ &= -\tfrac{1}{2}\tfrac{1}{2n-1} \left(\textstyle\sum_{x=0}^{2n} 2g(x) \sin \frac{\pi x}{2n} - \sum_{x=2}^{2n} g(x) \sin \frac{\pi(x-1)}{2n}\right. \\ &\quad \left. - \textstyle\sum_{x=0}^{2n-2} g(x) \sin \frac{\pi(x+1)}{2n}\right) \\ &= \tfrac{1}{2}\tfrac{1}{2n-1} \left(\textstyle\sum_{x=0}^{2n} g(x)(-2 \sin \frac{\pi x}{2n} + \sin \frac{\pi(x-1)}{2n} + \sin \frac{\pi(x+1)}{2n}\right) \end{aligned}$$

Nach den Additionstheoremen für den Sinus gilt:

$$\sin\left(\frac{\pi x}{2n} \pm \frac{\pi}{2n}\right) = \sin \frac{\pi x}{2n} \cos \frac{\pi}{2n} \pm \cos \frac{\pi x}{2n} \sin \frac{\pi}{2n}$$

Außerdem ist :

$$\cos x = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \ldots$$

Damit erhält man:

$$\begin{aligned} -2\sin\tfrac{\pi x}{2n} + \sin\tfrac{\pi(x-1)}{2n} + \sin\tfrac{\pi(x+1)}{2n} &= \sin\tfrac{\pi x}{2n}(-2+2\cos\tfrac{\pi}{2n}) \\ &= \sin\tfrac{\pi x}{2n}((\tfrac{\pi}{2n})^2 + \mathrm{O}(\tfrac{1}{n^4})) \end{aligned}.$$

Also ist

$$E[\Delta\Phi|X_t,Y_t] \approx -\frac{1}{2}\frac{1}{2n-1}\sum_{x=0}^{2n} g(x)\sin\frac{\pi x}{2n}(\frac{\pi}{2n})^2 = -\frac{1}{2}\frac{1}{2n-1}(\frac{\pi}{2n})^2\Phi(t).$$

□

Nach diesen Vorbereitungen kann nun die Mischzeit für die Markov-Kette bestimmt werden: Nach Satz 6.1 ist

$$E[\Phi(t)] \approx \Phi(0)(1-\frac{1}{2}\frac{1}{2n-1}(\frac{\pi}{2n})^2)^t \approx \Phi(0)e^{-\frac{\pi^2 t}{8n^2(2n-1)}}$$

Mit $\Phi(0) < n^2$ wird dieser Erwartungswert kleiner als der kleinste mögliche Wert der Funktion Φ, nämlich $\sin\frac{\pi}{2n}$, wenn

$$t > \frac{(2\log n - \log\sin\frac{\pi x}{2n})8n^2(2n-1)}{\pi^2} = \mathrm{O}(n^3\log n)$$

ist. Nach der Markov-Ungleichung genügen also mit hoher Wahrscheinlichkeit $\mathrm{O}(n^3\log n)$ Schritte, damit Φ einen Wert annimmt, der kleiner als ihr kleinster echt positiver Wert ist, also Null wird. Damit ist auch die Couplingzeit $\mathrm{O}(n^3\log n)$. Nach Satz 5.2 gilt dann für die Mischzeit der Markov-Kette

$$\tau(\varepsilon) \le 8T\log\frac{1}{\varepsilon} = \mathrm{O}(n^3\log n\log\frac{1}{\varepsilon})$$

Das Programm zur Erzeugung eines Dyck-Pfades mit Hilfe dieser Markov-Kette findet sich in Abschnitt A.4.

6.2 Eine andere Verteilung

Nicht immer soll eine Struktur nach der Gleichverteilung ausgewählt werden, deshalb soll an dieser Stelle auch eine andere Verteilung betrachtet werden. Für die Dyck-Pfade bietet sich eine Verteilung an, bei der ein einzelner Pfad mit einer Wahrscheinlichkeit proportional zu q^f erzeugt wird, wobei f die Fläche unter dem Pfad bezeichnet und $q \le 1$ einen geeigneten Parameter.

Die dafür benutzte Markov-Kette hat als Zustände wieder die Dyck-Pfade von $(0,0)$ nach $(0,2n)$. Einen Übergang von einem Dyck-Pfad X zu einem anderen

Y kann es genau dann geben, wenn sie sich nur durch einen Flip unterscheiden. Allerdings ändern sich jetzt die Wahrscheinlichkeiten:

$$P_{X,Y} = \begin{cases} \frac{1}{1+q^2}\frac{1}{2n-1} & \text{, falls } Y \text{ aus } X \text{ durch einen Abwärtsflip hervorgeht} \\ \frac{q^2}{1+q^2}\frac{1}{2n-1} & \text{, falls } Y \text{ aus } X \text{ durch einen Aufwärtsflip hervorgeht} \\ 1 - \sum_{Z \neq X} P_{X,Z} & \text{, falls } X = Y \\ 0 & \text{, sonst} \end{cases}$$

Simuliert wird ein Schritt dieser Kette wie oben, indem $p \in \{1 \dots 2n-1\}$ gleichverteilt und $r \in \{\uparrow, \downarrow\}$ mit Wahrscheinlichkeit $Pr(\uparrow) = \frac{q^2}{1+q^2}$ und $Pr(\downarrow) = \frac{1}{1+q^2}$ gewählt werden. Die Ausführung eines Schrittes erfolgt wie bereits beschrieben.

Diese Markov-Kette ist aperiodisch, da die Wahrscheinlichkeit in einem Zustand zu bleiben auch hier immer größer Null ist, woraus folgt, daß $\text{ggt}\{t | (P^t)_{ij} > 0\} = 1$ für alle Zustände i, j ist. Sie ist irreduzibel, da jeder Pfad von jedem anderen aus durch Flips erreicht werden kann. Also ist sie ergodisch.

Behauptung: Die stationäre Verteilung π ist die gewünschte Verteilung, also $\pi_X = c \cdot q^{f(X)}$ für alle Pfade X, wobei $f(X)$ die Fläche unter dem Pfad X bezeichnet. c ist eine Konstante, die sich daraus ergibt, daß π eine Wahrscheinlichkeitsverteilung ist, also die Summe 1 hat. Das heißt $c = \frac{1}{\sum_X q^{f(X)}}$.

Beweis: Zu zeigen ist, daß π die Bedingung $\pi = \pi P$ erfüllt: Sei f die Fläche unter dem Pfad X. Dann haben alle Pfade Y, die durch einen Aufwärtsflip aus X hervorgehen ($X \uparrow Y$), eine Fläche von $f+2$ und alle Pfade Z, die durch einen Abwärtsflip aus X hervorgehen ($X \downarrow Z$), eine Fläche von $f-2$. Das bedeutet:

$$\begin{aligned} (\pi P)_X &= \sum_{Y \downarrow X} cq^{f+2} \frac{1}{1+q^2}\frac{1}{2n-1} + \sum_{Z \uparrow X} cq^{f-2} \frac{q^2}{1+q^2}\frac{1}{2n-1} + cq^f(1 - \sum_{D \neq X} P_{X,D}) \\ &= cq^f(\frac{1}{1+q^2}\frac{1}{2n-1}(\sum_{Y \downarrow X} q^2 + \sum_{Z \uparrow X} 1) + 1) \\ &\quad -(\frac{1}{1+q^2}\frac{1}{2n-1}(\sum_{X \uparrow Y} q^2 + \sum_{X \downarrow Z} 1)) \\ &= cq^f \\ &= \pi_X \end{aligned}$$

Ebenso sieht man, daß die Kette reversibel ist.

Zur Bestimmung der Mischzeit $\tau(\varepsilon)$ wird nun wieder ein Coupling $(X_t, Y_t) \in D_n \times D_n$ definiert. In diesem Fall wird als Distanzfunktion $\varphi(X, Y)$ die Summe aus den Flächen zwischen den beiden Gitterpfaden X bzw. Y und dem minimalen Pfad P_m, nämlich dem Zickzack-Pfad von $(0,0)$ nach $(2n, 0)$, benutzt. Das bedeutet, φ nimmt Werte zwischen 0 und $2(n^2 - n)$ an. Zur Vereinfachung der Schreibweise sei $\Phi(t) = \varphi(X_t, Y_t)$. Wird der Abstand zwischen dem oberen Pfad X und diesem minimalen Pfad Null, dann muß auch der Abstand zum eigentlich betrachteten Pfad Y inzwischen Null geworden sein, d.h. das Coupling hat stattgefunden.

Es gilt das folgende Lemma:

Satz 6.2 *Falls Φ Werte zwischen 0 und höchstens B annimmt und $E[\Delta\Phi(t)|X_t, Y_t] \leq 0$ und $E[(\Delta\Phi(t))^2|X_t, Y_t] \geq V$, wann immer $\Phi(t) > 0$, dann ist die erwartete Coupling-Zeit für Startwerte X und Y $E[T^{X,Y}] \leq \Phi(0)\frac{2B-\Phi(0)}{V}$.*

Beweis: Sei $Z(t) = \Phi(t)^2 - 2B\Phi(t) - Vt$. Es gilt

$$\begin{aligned} E[Z(t+1)|Z(t), \ldots Z(0)] - Z(t) &= E[\Phi(t+1)^2 - 2B\Phi(t+1) - V(t+1)|\Phi(t)] \\ &\quad -\Phi(t)^2 + 2B\Phi(t) + Vt \\ &= E[\Phi(t)^2|\Phi(t)] + E[2\Phi(t)\Delta\Phi(t)|\Phi(t)] \\ &\quad +E[(\Delta\Phi(t))^2|\Phi(t)] - E[2B\Phi(t)|\Phi(t)] \\ &\quad -E[2B\Delta\Phi(t)|\Phi(t)] - Vt - V - \Phi(t)^2 \\ &\quad +2B\Phi(t) + Vt \\ &= 2(\Phi(t) - B)E[\Delta\Phi(t)|X_t, Y_t] \\ &\quad +E[(\Delta\Phi(t))^2|X_t, Y_t] - V \\ &\geq 0 \end{aligned}$$

$\Rightarrow Z$ ist Submartingal.
$T^{X,Y}$ ist eine Stopzeit für Z mit endlichen Erwartungswert, $|Z(t+1) - Z(t)|$ ist beschränkt für jedes t. Das heißt, das optionale Stop-Theorem für Submartingale kann angewendet werden:

$$E[Z(T^{X,Y})] \geq E[Z(0)]$$

$\Rightarrow$

$$\begin{aligned} E[\Phi(T^{X,Y})^2 - 2B\Phi(T^{X,Y}) - VT^{X,Y}] &= E[-VT^{X,Y}] \\ &= -VE[T^{X,Y}] \\ &\geq \Phi(0)^2 - 2B\Phi(0) \end{aligned}$$

Daraus folgt die Aussage des Satzes.

□

Um diesen Satz anwenden zu können, muß $E[\Delta\Phi]$ bestimmt werden. Es gilt:

Satz 6.3 *Seien X und Y Dyck-Pfad mit $X, Y \neq P_m$. Dann ist $E[\Delta\Phi|X, Y] \leq 0$.*

Beweis: Es sei $X \neq P_m$. Zuerst werden alle Spitzen auf dem Pfad X als *gut*, alle Täler als *schlecht* gekennzeichnet. Ein Punkt ist also gut, wenn ein Flip hier eine Flächenverkleinerung bedeutet, schlecht bei einer Flächenvergrößerung. Die betrachteten Punkte im Inneren des Segments sind abwechselnd mit gut und schlecht gekennzeichnet, wobei es auf dem Pfad einen Überschuß von einem guten Punkt geben muß. Das heißt:

$$\begin{aligned}
E[\Delta\Phi \text{bezüglich} X] &= (\text{Anzahl gute Punkte})\, c\frac{1}{1+q^2}\frac{1}{2n-1}\cdot(-2) \\
&\quad +(\text{Anzahl gute Punkte } -1)c\frac{q^2}{1+q^2}\frac{1}{2n-1}\cdot 2 \\
&= ((\text{Anzahl gute Punkte})\,(q^2-1)-q^2)c\frac{1}{1+q^2}\frac{1}{2n-1}\cdot 2 \\
&\leq 0
\end{aligned}$$

da $q^2 \leq 1$ ist.

Analog für Y. Insgesamt ist dann auch $E[\Delta\Phi|X,Y] \leq 0$.

□

Nach diesen Vorbereitungen kann nun die Mischzeit für die Markov-Kette bestimmt werden:

Satz 6.4 *Die Mischzeit für die beschriebene Markov-Kette auf den Dyck-Pfaden ist* $\tau(\varepsilon) = O(n^5 \log \frac{1}{\varepsilon})$.

Beweis: Nach Satz 5.2 genügt es zu zeigen, daß die Coupling-Zeit $T = \mathrm{O}(n^5)$ ist. Dazu wird Satz 6.2 benutzt. φ liefert Werte aus $[0..2(n^2-n)]$. Nach Satz 6.3 ist $E[\Delta\Phi] \leq 0$. Um Satz 6.2 anwenden zu können, muß also nur noch $E[(\Delta\Phi)^2]$ beschränkt werden, wenn die Flächen zwischen den Pfaden nicht Null sind. Es gibt also ein Segment, in dem ein Pfad echt über P_m ist. Es gibt also mindestens einen guten Punkt auf einem der beiden Pfade. Wenn dieser Punkt beim nächsten Schritt gewählt wird, gibt es eine Flächenverkleinerung mit Wahrscheinlichkeit $\geq \frac{q^2}{1+q^2}$. Φ verkleinert sich also echt mit einer Wahrscheinlichkeit $\geq \frac{q^2}{1+q^2}\frac{1}{2n-1}$.
$\Rightarrow E[(\Delta\Phi)^2] \geq 4 \cdot \frac{q^2}{1+q^2}\frac{1}{2n-1}$.
Nach Satz 6.2 gilt also:

$$\begin{aligned}
E[T^{X,Y}] &\leq \Phi(0)\frac{2B-\Phi(0)}{4\cdot\frac{q^2}{1+q^2}\frac{1}{2n-1}} \\
&\leq 2(n^2-n)(2\cdot 2(n^2-n)-2(n^2+n))\frac{(1+q^2)(2n-1)}{4q^2} \\
&= \mathrm{O}(n^5)
\end{aligned}$$

□

7 Erzeugung von Triangulierungen

7.1 Gleichverteilung

Auch für die Erzeugung von Triangulierungen gibt es eine natürliche lokale Operation, die den Schritt einer Markov-Kette definiert: den Flip einer Diagonalen, d.h. eine Diagonale der aktuellen Triangulierung wird ausgewählt, entfernt und durch die andere im entstehenden Viereck ersetzt. Diese Markov-Kette wird allgemein als *Triangulation Walk* bezeichnet und wird näher in [12] und [13] behandelt. Zur Abschätzung der Misch-Zeit wird hier der Vergleich mit der bereits behandelten Markov-Kette auf den Dyck-Pfaden benutzt.

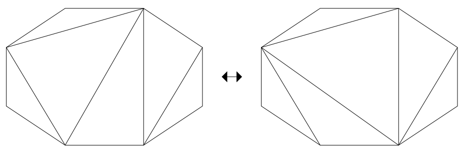

Abbildung 20: Ein Flip

Der durch die Flips definierte Graph auf den Triangulierungen ist isomorph zum Graph der binären Bäume mit der Rotations-Operation. Am leichtesten sieht man dies an der Zeichnung in Abbildung 21: Ein Flip der Diagonalen d entspricht gerade einer Rotation des Knotens, der zu d gehört. Zur Vereinfachung der Darstellung kann daher dieser Graph betrachtet werden.

Die Markov-Kette auf dem Baum-Graphen sei nun gegeben durch

$$P(x,y) = \begin{cases} \frac{1}{2(n-1)} & \text{, falls } y \text{ durch eine Rotation aus } x \text{ hervorgeht} \\ \frac{1}{2} & \text{, falls } x = y \\ 0 & \text{, sonst} \end{cases}$$

Für die Dyck-Pfade ergibt sich wie oben

$$\widetilde{P}(x,y) = \begin{cases} \frac{1}{2(2n-1)} & \text{, falls } y \text{ durch einen Flip aus } x \text{ hervorgeht} \\ 1 - \sum_{y \neq x} \widetilde{P}(x,y) \geq \frac{1}{2} & \text{, falls x=y} \\ 0 & \text{, sonst} \end{cases}$$

Beide Markov-Ketten sind reversibel und irreduzibel und haben die Gleichverteilung als stationäre Verteilung π bzw. $\widetilde{\pi}$. Nach der obigen Analyse ist die Mischzeit der Markov-Kette auf den Dyck-Pfaden $\widetilde{\tau}(\varepsilon) = \mathrm{O}(n^3 \log n \log \frac{1}{\varepsilon})$.

Angewendet werden soll Satz 5.3, der die Mischzeit auf der Baum-Kette mit Hilfe dieser bereits bekannten Mischzeit abschätzt. Die beiden Ketten arbeiten zwar nicht direkt auf dem gleichen Zustandsraum, aber da es eine Bijektion zwischen

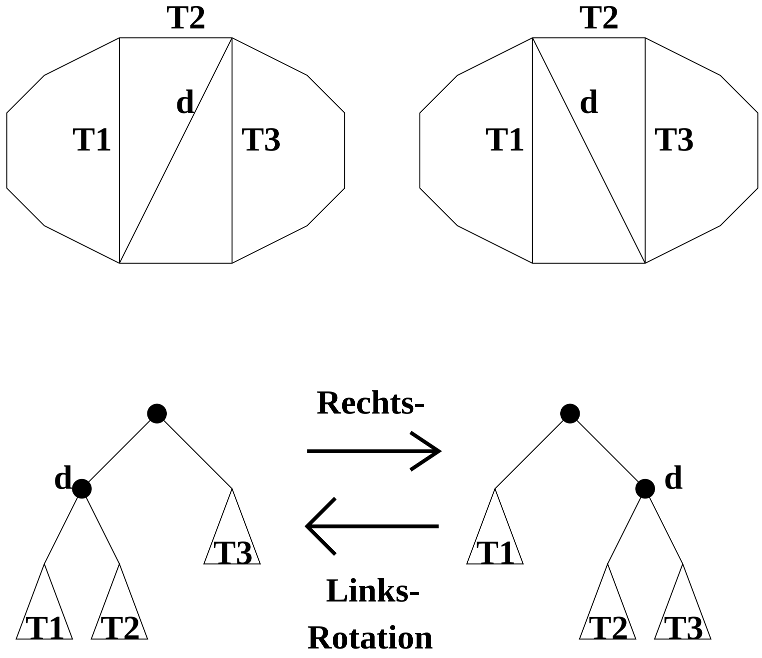

Abbildung 21: Isomorphie zwischen Baum-Rotationen und Diagonalen-Flips

den binären Bäumen und den Dyck-Pfaden gibt, läßt sich der Satz ebenso anwenden. Wie bereits gesagt wurde, haben beide Ketten die gleiche stationäre Verteilung. Es bleibt also zur Anwendung des Satzes die Festlegung der Pfade γ_{xy}. Seien also x und y Dyck-Pfade, die sich nur in einem Flip unterscheiden, z.B. y geht durch einen Aufwärtsflip aus x hervor. Seien $b(x)$ bzw. $b(y)$ die Bäume, die den Pfaden entsprechen. Gefunden werden muß nun eine Sequenz von Rotationen, die Baum $b(x)$ in Baum $b(y)$ umwandelt. Die folgende Beobachtung aus Abbildung 22 erlaubt die Festlegung einer solchen Sequenz: Ein Aufwärtsflip verschiebt einen linken Teilbaum, der an einem rechten Kind eines Knotens N hängt, so daß er der rechteste Teilbaum des linken Kindes von N wird. Der Pfad $\gamma_{x,y}$ besteht damit aus einer Rotation des Elternknotens des entsprechenden Teilbaums und dann Rotationen am Bruder der Wurzel des wandernden Teilbaums. Umgekehrt besteht der Pfad $\gamma_{y,x}$ so lange aus Rotationen des Vaters des wandernden Teilbaums, bis dieser ein linkes Kind ist, wenn y durch einen Abwärtsflip aus x hervorgeht.

Zur Anwendung des Satzes muß nun $A(\Gamma)$ beschränkt werden. Um dies zu erreichen, wird die Länge eines Pfades γ_{xy} betrachtet: Bei der ersten Rotation bleibt die Tiefe der Wurzel des zu verschiebenden Teilbaums gleich, bei jeder folgenden Rotation nimmt sie um 1 zu. Da die Tiefe höchstens $n-1$ mal vergrößert werden kann, hat der Pfad höchstens Länge n.

Betrachte weiterhin eine beliebige Rotation (v, w). Je nachdem, ob dies eine

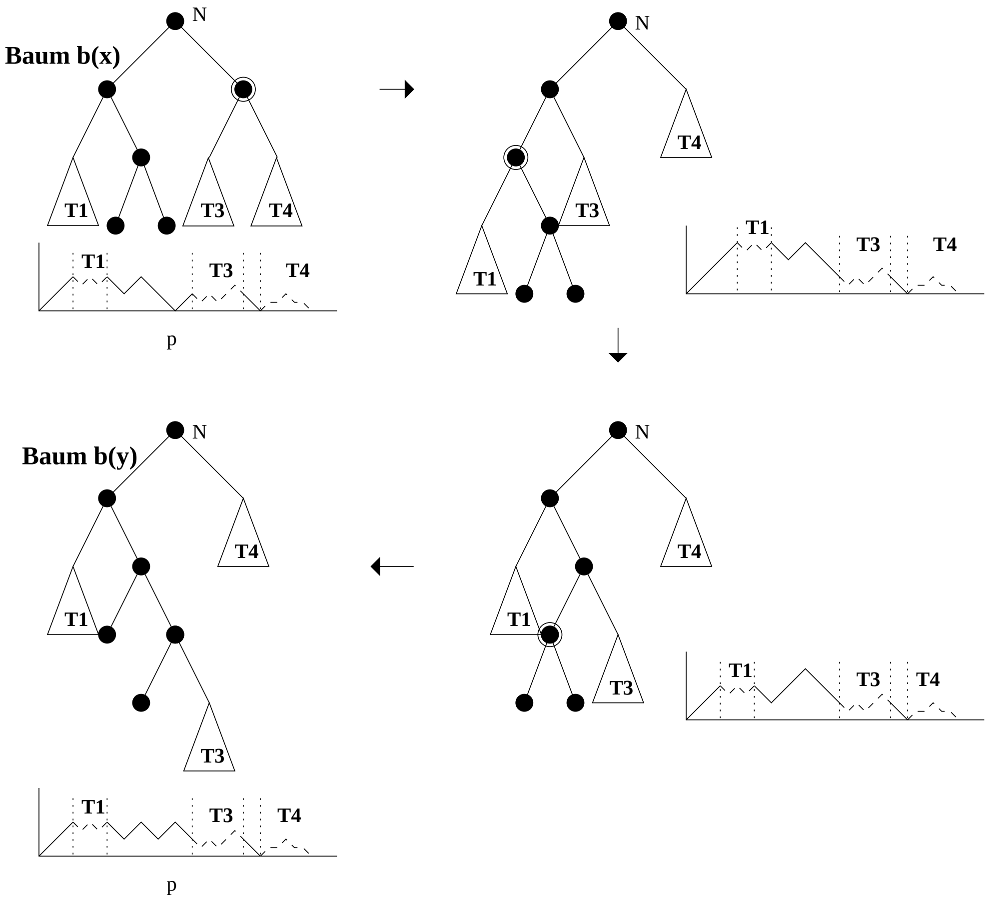

Abbildung 22: Verschiebung des Teilbaums T_3 beim Aufwärtsflip

Rechts- oder Links-Rotation ist, gibt es immer höchstens zwei Möglichkeiten, welcher Teilbaum der zu verschiebende ist. Wie in Abbildung 21 ist bei der Rechts-Rotation entweder T_3 der zu verschiebende Teilbaum eines Aufwärtsflips oder T_2 der zu verschiebende Teilbaum eines Abwärtsflips. Die Rotation (v, w) und die Wahl von T_3 oder T_2 als zu verschiebender Teilbaum bestimmen damit das Paar (x, y) mit $(v, w) \in \gamma_{xy}$ eindeutig. Analog für eine Links-Rotation. Zusammen folgt daraus, daß $|\Gamma(v, w)| \leq 2$ ist für alle Rotationen (v, w) und daß, wenn $(v, w) \in \gamma_{xy}$, $|\gamma_{xy}| \leq n$.

Mit $\pi_x = \frac{1}{C_n} = \frac{n+1}{\binom{2n}{n}}$ für alle x, $P(v, w) = \frac{1}{2n-2}$ für alle Rotationen (v, w) und $\widetilde{P}(x, y) = \frac{1}{4n-2}$ für alle Flips (x, y) erhält man:

$$\begin{aligned} A(\Gamma) &= \max_{P(v,w)>0}\{\tfrac{1}{\pi_v P(v,w)} \textstyle\sum_{\Gamma(v,w)} |\gamma_{xy}| \pi_x \widetilde{P}(x, y)\} \\ &\leq \tfrac{\binom{2n}{n} 2(n-1)}{n+1} \cdot 2n \tfrac{n+1}{\binom{2n}{n} 2(2n-1)} \\ &\leq n = \mathrm{O}(n) \end{aligned}$$

Mit Satz 5.3 kann nun die Mischzeit für die Markov-Kette auf den binären Bäumen bzw. auf den Triangulierungen abgeschätzt werden:

$$\tau(\varepsilon) \leq \frac{4\log\frac{1}{\varepsilon\pi_*}}{\log\frac{1}{2\varepsilon}} n \ \mathrm{O}(n^3 \log n \log\frac{1}{\varepsilon}) = \mathrm{O}(n^5 \log n \log\frac{n}{\varepsilon})$$

Bemerkung: Molloy, Reed und Steiger haben in [13] gezeigt, daß die Mischzeit nicht besser sein kann als $\Omega(n^{\frac{3}{2}})$.

7.2 Triangulierung mit fester längster Diagonale

Definition 7.1 *Für eine Diagonale v_iv_j mit $i > j$ ist die Länge $||v_iv_j|| = \min(i-j, n+2-i+j)$. Die Länge ist also die (kleinere) Anzahl aufeinanderfolgender Kanten von K zwischen den Endpunkten der Diagonale.*

Im binären Baum, der einer Triangulierung entspricht, ist jede Kante von K ein Blatt außer der Kante e_0, die zur Wurzel wurde (vergleiche Kapitel 2). Die Länge einer Diagonalen ist demnach also gleich der Anzahl der Blätter im Teilbaum mit dem Wurzelknoten, der der Diagonale entspricht, oder der Anzahl der Blätter im restlichen Baum $+1$ für die Wurzelkante e_0.

Definition 7.2 *Für eine Triangulierung t sei $l_n(t) = \max_{v_iv_j \in t} ||v_iv_j||$ die Länge der längsten Diagonalen von t.*

Im entsprechenden binären Baum ist die Länge der längsten Diagonalen die Anzahl der Blätter im größten Teilbaum, der nicht mehr als die Hälfte der Blätter enthält.

Klar ist:

$$\frac{n+2}{3} \leq l_n(t) \leq \frac{n+2}{2}$$

Sei also $\frac{n+2}{3} \leq k \leq \frac{n+2}{2}$ die gewünschte Länge der längsten Diagonale. Zu Beginn des Algorithmus' wird die Diagonale v_0v_k generiert, die diese Länge hat. Generiert werden muß nun ein i, um das Dreieck $v_0v_kv_i$ zu vervollständigen. Um zu gewährleisten, daß keine der weiteren Diagonalen eine größere Länge als k hat, muß i aus dem Bereich $[n+2-k, 2k]$ sein bzw. aus dem Bereich $[n+2-k+1, 2k-1]$, falls $k = \frac{n+2}{2}$ ist. Ist v_i die dritte Ecke zum Dreieck $v_0v_kv_i$, dann gibt es $C_{i-k-1}C_{n+2-i-1}$ Triangulierungen des Teilpolygons rechts von v_0v_k. Insgesamt sind bei allen möglichen Positionen von i also $S = \sum_{i=n+2-k}^{2k} C_{i-k-1}C_{n+2-i-1}$ bzw. $S = \sum_{i=n+2-k+1}^{2k-1} C_{i-k-1}C_{n+2-i-1}$ Triangulierungen rechts von v_0v_k möglich. Das bedeutet, der Knoten v_i wird mit Wahrscheinlichkeit $\frac{C_{i-k-1}C_{n+2-i-1}}{S}$ als dritter des Dreiecks ausgewählt. Um die Teilpolygone $(v_0 \dots v_k)$, $(v_k \dots v_i)$ und $(v_i \dots v_0)$ zu triangulieren, wird einer der beschriebenen Algorithmen zur zufälligen Erzeugung einer Triangulierung benutzt. Am Ende des Algorithmus' wird K rotiert, so daß die längste Diagonale an jedem Punkt beginnen kann, also $v_j v_{j+k}$ mit Wahrscheinlichkeit $\frac{1}{n+2}$ längste Diagonale ist.

1. Wähle i mit Wahrscheinlichkeit $\frac{C_{i-k-1}C_{n+2-i-1}}{S}$ aus.
2. Trianguliere $(v_0, \dots v_k)$, $(v_{k+1} \dots v_i)$ und $(v_{i+1} \dots v_0)$ zufällig.
3. Rotiere um ein zufällig gewähltes $I \in [0, n+1]$.

7.3 Triangulierung mit fester Anzahl von Ohren

Definition 7.3 *Eine Ecke v_i von K ist ein Ohr in der Triangulierung t, wenn $v_{i-1}v_{i+1} \in t$ ist.*

Sei $2 \le k \le \frac{n+2}{2}$ die gewünschte Anzahl von Ohren. Zuerst wird gezeigt, daß es $\frac{n+2}{k} \cdot 2^{n+2-2k} \begin{pmatrix} n-2 \\ n+2-2k \end{pmatrix}$ Triangulierungen mit k Ohren gibt. Der Algorithmus zur Erzeugung einer Triangulierung ergibt sich dann aus diesem Beweis.
Sei t eine Triangulierung mit Ohren in den Ecken $v_{j_1}, \dots v_{j_k}$. Es sei $j_1 = 0$. t hat $n-1$ Diagonalen, darunter die k Ohrendiagonalen $v_{j_i-1}v_{j_i+1}$. t wird nun kollabiert, indem alle Polygonkanten von K, die nicht zu einem Ohr gehören, entfernt werden, d.h. $v_i v_{i+1}$ wird entfernt und zu einer Ecke zusammengezogen, wenn in beiden kein Ohr ist. Die beiden Diagonalen, mit denen $v_i v_{i+1}$ ein Dreieck bildet, werden dabei zu einer einzigen zusammengefaßt. Es werden also $n+2-2k$ Kanten und ebenso $n+2-2k$ Diagonalen entfernt.

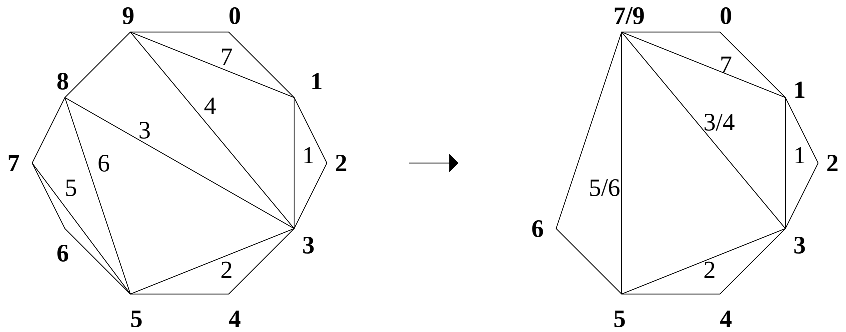

Abbildung 23: Kollabierung von K

Übrig bleibt das $2k$-gon K' mit der Triangulierung t'. Die Ohrendiagonalen bilden im Inneren von K' ein k-gon, für das es C_{k-2} Triangulierungsmöglichkeiten gibt. Um die Triangulierungen t zu zählen, die alle zu der gleichen Triangulierung t' kollabieren, nummeriere die Diagonalen von t, so daß $v_iv_j < v_iv_r \Leftrightarrow i < j < r$ und $v_iv_j < v_rv_s \Leftrightarrow i < r$. Die Diagonale v_1v_{n+1} erhält die Nummer $n-1$. Sie bleibt in t' in jedem Fall erhalten, aber $n+2-2k$ der anderen $n-2$ Diagonalen werden entfernt. Es gibt also $\binom{n-2}{n+2-2k}$ Möglichkeiten, diese auszuwählen, bzw. umgekehrt $\binom{n-2}{n+2-2k}$ Möglichkeiten, die entfernten Diagonalen wieder einzufügen. Soll eine Diagonale v_iv_j in t' nun wieder erweitert werden, gibt es dafür zwei Möglichkeiten, nämlich durch die Einführung einer neuen Ecke v_{i-1} und der Diagonalen $v_{i-1}v_j$ (Variante 1) oder durch die Enführung einer neuen Ecke v_{j-1} und der Diagonale v_iv_{j-1} (Variante 2). Damit gibt es also $2^{n+2-2k}\binom{n-2}{n+2-2k}$ Triangulierungen mit $v_{j_1} = v_0$. Nun kann natürlich jede Ecke von K die Rolle von v_0 übernehmen, indem die Triangulierung rotiert wird. Man erhält also noch einmal den Faktor $\frac{n+2}{k}$ für die Rotation rund um K, wobei jede Triangulierung k mal gezählt wird.

Algorithmus:

1. Trianguliere $(v_0 \dots v_{k-1})$ zufällig.
2. Füge k Ohren hinzu, d.h. $v_i := v_{2i+1}$ für $i = 1..k$ und zusätzliche Ecken v_{2i} für $i = 1..k$, das ergibt die Triangulierung t'.
3. Wähle zufällige $(2k-4)$-elementige Teilmenge S von $\{1 \dots n-2\}$ als Nummern der Diagonalen von t',
   bilde t durch Hinzufügen der Diagonalen zu den Nummern $\{1 \dots n-2\}\backslash S$ zufällig nach Variante 1 oder 2.
4. Rotiere um ein zufällig gewähltes $I \in [0, n+1]$.

7.4 Triangulierung mit festem Maximalgrad

Definition 7.4 *Der Grad d_i einer Polygonecke v_i in einer Triangulierung t ist die Anzahl der Diagonalen von t, die inzident zu v_i sind. Der Maximalgrad $\Delta_n(t)$ ist*

$$\Delta_n(t) = \max_{1 \le i \le n} d_i.$$

Es ist leicht zu sehen, daß die Sequenz der Grade die Triangulierung bestimmt. Benutzt man die Polygonkante $\overline{1(n+2)}$ als Wurzelkante, erhält man den Baum, der der Triangulierung entspricht. Es ist leicht zu sehen, daß der Grad der Ecke 1 in der Triangulierung gleich der Anzahl von Knoten auf dem Pfad von der Wurzel bis zum am weitesten links gelegenen Blatt im Baum ist ohne die Wurzel und das Blatt selbst. Wandelt man den Baum weiter in eine Klammersequenz um, wird der Pfad von der Wurzel zum Blatt eine Folge von öffnenden Klammern. Das heißt, die Triangulierung hat genau dann Grad k in der Ecke 1, wenn die entsprechende Klammersequenz mit $k+1$ öffnenden Klammern beginnt.

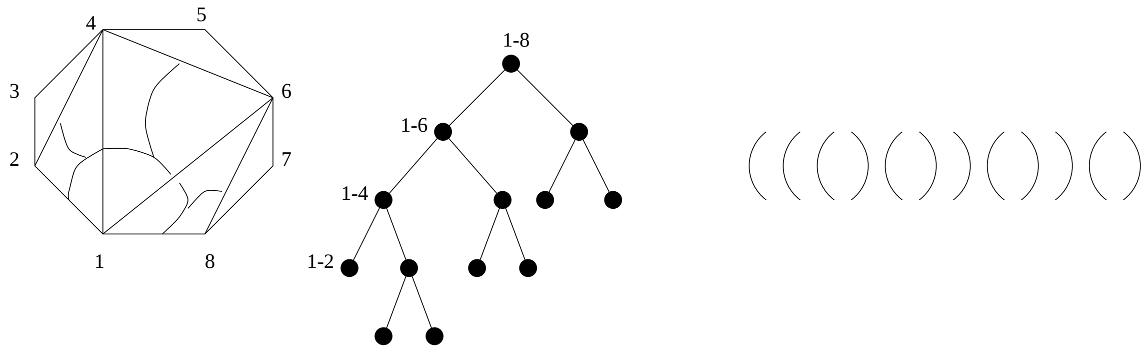

Abbildung 24: Der Grad einer Ecke und die entsprechende Klammersequenz

Das heißt, man erzeugt nicht direkt eine Triangulierung mit Maximalgrad k, sondern eine Klammersequenz, die $k+1$ aufeinanderfolgende öffnende Klammern enthält. Das Problem ist, daß es mehrere Polygonecken geben kann, die den Maximalgrad haben. Dadurch ist es schwierig, die Anzahl der möglichen Triangulierungen bzw. der Sequenzen zu bestimmen. Das bedeutet, daß auch die Wahrscheinlichkeiten, nach denen eine Triangulierung gewählt wird, nur schwer allgemein zu berechnen sind.

Ist allerdings $k \geq \frac{n-1}{2}$, soll also eine Polygonecke mindestens zur Hälfte der Diagonalen inzident sein, fällt dieses Problem weg. In diesem Fall sieht der Algorithmus wie folgt aus:

Algorithmus

1. Beginne eine Klammersequenz mit $k+1$ λ.
2. Berechne die restliche Sequenz mit den Wahrscheinlichkeiten aus dem Algorithmus von Arnold und Sleep (siehe 3.2).
3. Wandle die entstehende Klammersequenz in eine Triangulierung um.
4. Rotiere die Triangulierung um ein zufällig gewähltes $I \in [0, n+1]$.

Interessant ist die Erzeugung einer Triangulierung mit festem Maximalgrad k deshalb, weil eine solche einem Baum entspricht, in dem zwei aufeinanderfolgende Blätter höchstens den Abstand k haben. In einem Baum, der Informationen nur in den Blättern enthält, erleichtert dies den Durchlauf durch die Blätter.

8 Die Algorithmen in der Praxis

Um die einzelnen Algorithmen auch praktisch in ihrer Güte beurteilen zu können, wurden sie in Maple V programmiert. Die ermittelten Zeiten sind in $\frac{1}{100}$-Sekunden angegeben und sind natürlich abhängig von dem jeweils benutzten Rechner etc. Trotzdem liefern sie eine Möglichkeit, die Laufzeiten der einzelnen Algorithmen zu vergleichen.
Die behandelten Catalanobjekte werden folgendermaßen dargestellt:

1. Ein Dyck-Pfad ist eine Tabelle der Länge $2n + 1$, die die jeweiligen Höhen des Pfades angibt.

2. Eine Klammersequenz ist eine Tabelle der Länge $2n$ aus 1 und 2, wobei 1 eine öffnende Klammer, 2 eine schließende darstellt.

3. Ein Baum wird durch die Liste aus dem linken und dem rechten Teilbaum dargestellt, wobei eine leere Liste für ein Blatt steht.

4. Eine Triangulierung ist eine Tabelle aus den Diagonalen, aus denen die Triangulierung besteht. Die Ecken des zugrundeliegenden Polygons sind dazu von 1 bis $n + 2$ durchnummeriert. Eine Diagonale besteht aus den Tabelleneinträgen für ihre beiden Endpunkte.

Die Wahl von Tabellen zur Darstellung der einzelnen Strukturen erlaubt in Maple einen Zugriff auf die einzelnen Einträge in konstanter Zeit, die also nicht abhängig ist von der Größe der Tabelle. Nur so sind die gemachten Aussagen über die Laufzeiten der einzelnen Algorithmen überhaupt einzuhalten. Leider wird es dadurch insbesondere in den rekursiven Algorithmen nötig, jeweils den Beginn der noch freien Tabellenplätze anzugeben. Teilweise geht dabei etwas die Einfachheit bzw. Übersichtlichkeit der Algorithmen verloren.

8.1 Vor- und Nachteile der einzelnen Algorithmen

Die theoretische Betrachtung der Algorithmen ergab fünf mit einer linearen Laufzeit, während die beiden Markov-Ketten-Algorithmen eine kubische haben. Daraus könnte man schließen, daß die ersten fünf alle gleichwertig sind, während die letzten beiden überhaupt nicht eingesetzt werden sollten. In der Praxis sieht das ein wenig anders aus, wie die folgende Tabelle mit den durchschnittlichen Laufzeiten für verschiedene Eingabegrößen zeigt.

	arnsleep	atksack	knott	ruskeyhu
$n = 5$	0,002	0,003	0,002	0,002
$n = 10$	0,004	0,008	0,005	0,007
$n = 100$	0,048	0,082	0,125	0,595
$n = 500$	0,255	0,452	3,734	107,867
$n = 1000$	0,527	1,053	24,717	1150,816
$n = 2000$	1,107	2,648	242,883	11181,034

	baumwachsen	dyck_pfade	zufaellige_triangulierung
$n = 5$	0,008	0,116	62,733
$n = 10$	0,018	5,133	2963,543
$n = 100$	1,305	11275,797	$5,9 \cdot 10^8$ (**)
$n = 500$	12,766	1,8 $\cdot 10^6$	$2,7 \cdot 10^13$ (**)
$n = 1000$	47,734		
$n = 2000$	*		

Durchschnittliche Laufzeiten der einzelnen Algorithmen

(**) theoretische Werte

Besonders auffällig ist, daß die Algorithmen `knott`, `ruskeyhu` und `baumwachsen` eindeutig kein lineares Laufzeitverhalten zeigen, was den theoretischen Analysen widerspricht. Das liegt daran, daß diese Algorithmen mit den Catalanzahlen rechnen, die sehr groß werden können. Um z.B. die Eingabe $n = 2000$ verarbeiten zu können, wird im Algorithmus mit $C_{2000} \approx 10^{1199}$ gerechnet. Um eine solch große Zahl zu verarbeiten, ist wesentlich mehr Zeit erforderlich als eine lineare Laufzeit in n. Auf anderen Systemen, die nicht unbedingt für mathematische Anwendungen konzipiert wurden wie Maple, ist ein Umgang mit Zahlen dieser Größenordnung gar nicht bzw. nur unter sehr großem zusätzlichem Aufwand möglich.

Ein weiterer Nachteil dieser drei Algorithmen ist die Tatsache, daß die Tabellen für die Catalanzahlen bzw. die Werte der Funktion L erstens vorher berechnet werden müssen, was ebenfalls vergleichsweise lange dauert, und daß diese Werte auch gespeichert werden müssen, was bei der Größe der Zahlen sehr viel Speicherplatz erfordert. Die Tabelle für die Funktionswerte von L für $n = 2000$ war deswegen in Maple nicht mehr berechenbar. Dadurch bedingt, war auch der Baumgenerator `baumwachsen` für einen Baum mit 2000 inneren Knoten nicht mehr benutzbar.

	catalanliste	L_liste
$n = 5$	<0,001	<0,001
$n = 10$	<0,001	<0,001
$n = 100$	<0,001	0,483
$n = 500$	0,083	93,367
$n = 1000$	0,267	1343,850
$n = 2000$	2,883	*

Laufzeiten für die Berechnung der beiden Listen

In der obigen Tabelle wurden nur die Laufzeiten betrachtet, die für die Erzeugung der Struktur nötig sind, für die der jeweilige Algorithmus konzipiert wurde. Für die Umwandlung einer Struktur in eine andere wird zusätzliche Zeit benötigt. Das zeigt einen weiteren Nachteil der Baum-Algorithmen: Sie alle erzeugen nicht direkt einen Baum in der üblichen Form mit Wurzel, linkem und rechtem Teilbaum, sondern eine Darstellung des Baumes als Liste. Dadurch ist in den meisten Fällen eine Umwandlung der erzeugten Datenstruktur nötig. Insbesondere in `baumwachsen` ist dies ein Nachteil, da zur Umwandlung der Tabelle mit den zu erweiternden Endblättern in einen Baum in seiner normalen Darstellungsform eine Struktur benötigt wird, in die an jeder Stelle eingefügt werden kann. In Maple bieten sich dafür Listen an. Leider haben sie den Nachteil, daß ein Zugriff nicht in konstanter Zeit erfolgt, sondern linear in der Länge der Liste. dadurch erhöht sich die theoretische Laufzeit von `baumwachsen` um eine Potenz. Aber auch bei den anderen Algorithmen ist dieser zusätzliche Zeitverbrauch mit in die Überlegungen einzubeziehen. Anhand der obigen Tabelle hat `arnsleep` die beste Laufzeit, allerdings nur, wenn tatsächlich eine Klammersequenz erzeugt werden soll. Muß diese erst noch in eine andere Struktur umgewandelt werden, kann dieser Vorteil verloren gehen. Insbesondere eine Umwandlung einer Triangulierung in einen Baum sollte vermieden werden, da diese durch die Zuordnung der Diagonalen zum jeweiligen rechten oder linken Teilbaum eine quadratische Laufzeit hat.

	$K_n \to D_n$	$D_n \to B_n$	$B_n \to T_n$	$B_n \to K_n$	$T_n \to B_n$
$n = 5$	<0,001	0,001	0,001	<0,001	0,005
$n = 10$	0,001	0,001	0,002	<0,001	0,013
$n = 100$	0,006	0,012	0,025	0,006	0,404
$n = 500$	0,038	0,068	0,142	0,38	6,767
$n = 1000$	0,085	0,137	0,305	0,080	20,747
$n = 2000$	0,183	0,308	0,692	0,173	76,760

Laufzeiten für die Umwandlungsalgorithmen

Bei der zufälligen Erzeugung von Objekten ist nicht nur die Laufzeit wichtig, sondern auch die Zufallseigenschaften der Generatoren. Keiner der Algorithmen zeigte Auffälligkeiten beim Coupon-Collector-Test oder bei der χ^2-Metrik. Das ist vor allem darauf zurückzuführen, daß der in Maple benutzte Zufallszahlengenerator im Sinne dieser beiden Tests tatsächlich Zufallszahlen liefert. Bei der Implementierung muß also darauf geachtet werden, daß die Algorithmen nur so gut sein können wie der benutzte Zufallszahlengenerator. In den Algorithmen `arnsleep` und `baumwachsen` muß eine Zahl mit einer gegebenen Wahrscheinlichkeit erzeugt werden, die nicht unbedingt eine Gleichverteilung ist. Muß dies mit einem Generator simuliert werden, der gleichverteilt Zahlen aus einem festgelegten Bereich liefert, muß dieser Bereich mindestens für `baumwachsen` sehr groß sein, nämlich in der Größenordnung von C_n. Für die anderen Algorithmen dagegen sind Generatoren ausreichend, die Zufallszahlen zwischen 1 und n bzw. $2n$ für `atksack` liefern.

Ein weiteres Problem tritt beim Markov-Ketten-Algorithmus für die Triangulierungen auf: Für $n = 2$ gibt es nur eine Diagonale im $(n+2)$-gon, die also für jeden Flip ausgewählt wird. Dadurch wird das Ergebnis des Algorithmus vorhersagbar. Da die Auswahl aus den beiden möglichen Triangulierungen eine Vierecks auch leichter zu bewerkstelligen ist als mit Hilfe einer Markov-Kette, tritt dieses Problem wahrscheinlich bei der eigentlichen Erzeugung zufälliger Triangulierungen nicht auf. Allerdings liefert es einen Grund, in den Algorithmen zur Erzeugung zufälliger Triangulierungen mit fester Zahl von Ohren bzw. fester längster Diagonale `zufällige_triangulierung` nicht zu benutzen.

Zusammenfassend haben die einzelnen Algorithmen folgende Vor- und Nachteile:

Algorithmus	Laufzeit	Objekt	Vorteile	Nachteile
`arnsleep`	O(n)	Klammern	beste Laufzeit, Speicher in der Größe der erzeugten Struktur	
`atksack`	O(n)	Klammern		zusätzlicher Speicher für die zufällige Menge
`knott`	O(n)	Bäume	liefern andere Darstellung von Bäumen	für großes n sehr langsam bzw. undurchführbar, viel Speicher für C_n bzw. $L(j,n)$, Umwandlung in normale Baumdarstellung ist immer nötig
`ruskeyhu`	O(n)	Bäume		
`baumwachsen`	O(n)	Bäume		
`dyck_pfade`	$O(n^3 \log n)$	Dyck-Pfade	Speicher in der Größe der erzeugten Struktur, Laufzeitkonstante sehr klein	
`zuf_tria`	$O(n^5 \log n)$	Triangulierungen	Laufzeitkonstante sehr klein	nicht für $n = 2$ benutzbar

A Die Programme

Benutzt wurden diese beiden Hilfsprozeduren:

```
teil := proc(liste,von,bis)
```
berechnet liste[von..bis]
```
if (von <= bis) then RETURN(liste[von..bis]);
else RETURN(op([])); fi;
end:
```

```
zufaellig := proc(n,k)
```
erzeugt eine zufällige k-elementige Teilmenge von $\{1..n\}$
```
local i, menge, hilf, rn, j;
hilf := table();
for i from 1 to n do hilf[i]:=i; od;
menge := {};
for i from 1 to k do
  rn := rand(1..n-i+1);
  j := rn();
  menge := {op(menge),hilf[j]};
  hilf[j] := hilf[n-i+1]; od;
RETURN(menge);
end:
```

```
catalanliste := proc(n)
```
berechnet eine Tabelle der Catalanzahlen von 0 bis n mit Hilfe der Rekursionsformel
```
local i, tabelle;
tabelle := table([(0)=1]);
for i from 1 to n do
  tabelle[i] := (4*i-2)/(i+1)*tabelle[i-1];od;
RETURN(eval(tabelle));
end:
```

A.1 Umwandlung der einzelnen Catalanobjekte

A.1.1 Umwandlung einer Klammersequenz in einen Pfad:

```
pfad_aus_klammern := proc(klammern, n)
```
berechnet den der Klammersequenz entsprechenden Dyck-Pfad
```
local i, pfad;
pfad := table([(0)=0]);
for i from 1 to 2*n do
```

```
  if klammern[i]=1 then pfad[i] := pfad[i-1]+1;
```
Aufwärtsschritt
```
  else pfad[i] := pfad[i-1]-1; fi; od;
```
Abwärtschritt
```
RETURN(eval(pfad));
end:
```

A.1.2 Umwandlung eines Baums in eine Klammersequenz:

```
klammern_aus_baum:=proc(baum,n)
```
berechnet die dem Baum entsprechende Klammersequenz
```
local sequenz;
sequenz := table();
k_a_b(baum,sequenz,1);
RETURN(eval(sequenz));
end:
```

```
k_a_b := proc(baum, sequenz, i)
```
schreibt die dem Baum entsprechende Klammersequenz in sequenz, beginnend bei Eintrag i, Ausgabe ist der Index der nächsten zu beschreibenden Stelle in sequenz
```
local j;
if baum =[] then RETURN(i); fi;
sequenz[i] := 1;
```
Beginn eines linken Teilbaums
```
j := k_a_b(baum[1],sequenz,i+1);
sequenz[j] := 2;
```
daran anschließend der rechte Teilbaum
```
j := k_a_b(baum[2],sequenz,j+1);
RETURN(j);
end:
```

A.1.3 Umwandlung eines Baums in eine Triangulierung:

```
triangulierung_aus_baum := proc(baum, n)
```
berechnet die dem Baum entsprechende Triangulierung
```
local blaetter, t, t2, j, i;
```
blaetter : Liste, wieviele Blätter sich in den einzelnen Teilbäumen von baum befinden
t: Triangulierung, die außer den Diagonalen auch noch die Kanten des Polygons enthält
t2: die eigentliche Triangulierung

```
blaetter := blaetterliste(baum);
t := table();
t_a_b(blaetter,1,n+2,t,1);
```
die übergebene Kante $\overline{1(n+2)}$ *ist die Wurzelkante der zu berechnenden Triangulierung*
```
t2 := table();
```
aus t werden alle Polygonkanten gestrichen:
```
j := 1;
for i from 1 to 2*n do
  if t[i,2]-t[i,1] >1 then
    t2[j,1] := t[i,1]; t2[j,2] := t[i,2]; j := j+1; fi; od;
RETURN(eval(t2));
end:
```

Eine Triangulierung hat die Kanten $\overline{p1(p1+i)}$ und $\overline{(p1+i)p2}$, falls der linke Teilbaum mit Wurzelkante $\overline{p1p2}$ i Blätter enthält. Nach diesem Prinzip arbeitet die Umwandlungsprozedur `t_a_b`:

```
t_a_b := proc(blaetter,p1,p2,t,a)
```
berechnet aus der blaetterliste eines Baums die zugehörige Triangulierung mit Wurzelkante $\overline{p1p2}$ *und schreibt sie in t, beginnend an Stelle a*
```
local i;
if blaetter <> [[1,0]] then
  t := table([(a,1)=p1,(a,2)=p1+blaetter[1][1],
              (a+1,1)=p1+blaetter[1][1],(a+1,2)=p2]);
```
die beiden neuen Kanten eintragen
```
  t_a_b(blaetter[2],p1,p1+blaetter[1][1],t,a+2);
```
$\overline{p1(p1+blaetter[1][1])}$ *ist die neue Wurzelkante der Triangulierung zum linken Teilbaum*
```
  t_a_b(blaetter[3],p1+blaetter[1][1],p2,t,a+2*blaetter[1][1]);
```
genauso für den rechten Teilbaum
```
  RETURN();
else RETURN(); fi;
end:
```

```
blaetterliste := proc(baum)
```
berechnet eine Liste, wieviele Blätter sich jeweils in den Teilbäumen befinden. Die Liste ist von der Form [[Anz. Blätter im linken Teilbaum, Anz. Blätter im rechten Teilbaum], blaetterliste(linker Teilbaum), blaetterliste(rechter Teilbaum)]
[[1,0]] entspricht dabei einem Blatt
```
local bl, br;
```
bl, br: blaetterliste des linken bzw. rechten Teilbaums
```
if baum = [] then RETURN([[1,0]]); fi;
bl := blaetterliste(baum[1]);
```

```
br := blaetterliste(baum[2]);
RETURN([[bl[1][1]+bl[1][2], br[1][1]+br[1][2]], bl, br]);
end:
```

A.1.4 Umwandlung einer Triangulierung in einen Baum:

```
baum_aus_triangulierung := proc(t)
```
berechnet einen Baum aus der Triangulierung t
```
local t_lang, i;
```
t_lang: die um die Polygonkanten erweiterte Triangulierung
```
t_lang := copy(t);
```
Polygonkanten hinzufügen:
```
for i from 1 to n+1 do
  t_lang[i+n-1,1] := i; t_lang[i+n-1,2] := i+1; od;
t_lang[2*n+1,1] := 1; t_lang[2*n+1,2] := n+2;
RETURN(b_a_t(t_lang,n+2,1,2*n+1));
```
die erste Wurzelkante ist $\overline{(n+2)1}$
```
end:
```

```
b_a_t := proc(t,p1,p2,anz)
```
berechnet den Baum zur Triangulierung t mit Wurzelkante $\overline{p1p2}$
Voraussetzung: p1 ist im Uhrzeigersinn betrachtet rechts von p2, t enthält unter den insgesamt anz Einträgen auch die Polygonkanten
```
local dritterr, dritterl, index, tlinks, trechts, i,anzr,anzr;
```
dritterr, dritterl: der dritte Punkt des Dreiecks zu $\overline{p1p2}$*, von rechts, also von p1 aus gesucht, bzw. von links, also von p2 aus*
tlinks, trechts: Teiltriangulierungen für den linken bzw. rechten Teilbaum
anzl, anzr: Anzahl Einträge in den Teiltriangulierungen
```
if anz=0 then RETURN([]); fi;
```
Finden des dritten Dreieckpunktes:
```
index := 1;
dritterr := p1-1;
dritterl := p2+1;
while dritterr > dritterl do
  if member(p1, {t[index,1],t[index,2]}) and
     dritterr > op({t[index,1],t[index,2]} minus {p1}) then
```
p1 ist einer der Punkte der gerade betrachteten Kante, der andere ist links vom bekannten dritterr
```
    dritterr := op({t[index,1],t[index,2]} minus {p1}); fi;
  if member(p2, {t[index,1],t[index,2]}) and
     dritterl < op({t[index,1],t[index,2]} minus {p2}) then
```
p2 ist einer der Punkte der gerade betrachteten Kante, der andere ist rechts von

dritterl

```
    dritterl :=op({t[index,1],t[index,2]} minus {p2}); fi;
  index := index +1; od;
```

Zuordnung der restlichen Diagonalen und Polygonkanten zum linken bzw. rechten Teil

```
tlinks := table(); anzl := 0;
trechts := table(); anzr := 0;
for i from 1 to anz do
  if t[i,1] >= p2 and t[i,2] >= p2 and t[i,1]<=dritterr and
     t[i,2]<=dritterl and {t[i,1],t[i,2]})} <> {p2,dritterl} then
    anzl := anzl+1;
    tlinks[anzl,1] := t[i,1]; tlinks[anzl,2] := t[i,2];
  else
    if t[i,1]>=dritterr and t[i,2] >= dritterr and
       {t[i,1],t[i,2]}<>{p1,dritterr} then
      anzr := anzr +1;
      trechts[anzr,1] := t[i,1];
      trechts[anzr,2] := t[i,2]; fi; fi; od;
RETURN([b_a_t(tlinks,dritterr,p2,anzl),
        b_a_t(trechts,p1,dritterr,anzr)]);
```

der Ausgabebaum setzt sich aus den Teilbäumen, die die anderen beiden Kanten des Dreiecks zu $\overline{p1p2}$ als Wurzel haben, zusammen

```
end:
```

A.1.5 Umwandlung eines Pfads in einen Baum:

```
baum_aus_pfad:= proc(pfad,n)
```

berechnet einen Baum aus dem Dyck-Pfad pfad

```
RETURN(b_a_p(pfad,1,n)[1]);
```

in der Rückgabeliste von t_a_p steht der eigentliche Baum an erster Stelle

```
end:
```

```
b_a_p:= proc(pfad,i,n)
```

baut den zu pfad ab Stelle i gehörigen Baum mit Hilfe einer Tiefensuche auf, die Rückgabe hat die Form [Baum, Beginn des noch zu verarbeitenden Restpfads]

```
local baum, links, rechts;
```

links, rechts: linker bzw. rechter Teilbaum mit verbliebenem Restpfad

```
if i>2*n or pfad[i]<pfad[i-1] then RETURN([[],i]); fi;
```

ein Abwärtsschritt bedeutet, daß in pfad ein neuer rechter Teilbaum beginnt

```
links := b_a_p(pfad,i+1,n);
rechts := b_a_p(pfad,links[2]+1,n);
baum := [[links[1],rechts[1]],rechts[2]];
```

der Rückgabebaum setzt sich aus den beiden Teilbäumen und dem Beginn des noch verbliebenen Restpfades zusammen

```
RETURN(baum);
end:
```

A.2 Erzeugung von gültigen Klammersequenzen

A.2.1 Algorithmus von Atkinson und Sack

```
atksack := proc(n)
```

erzeugt eine zufällige Klammersequenz aus n Klammerpaaren mit dem Algorithmus von Atkinson und Sack

```
local k,menge,i,sequenz;
```

k: balancierte Klammersequenz
menge: n-elementige Teilmenge von $\{1..2n\}$
sequenz: die zu erzeugende gültige Sequenz

```
menge := zufaellig(2*n,n);
k:=table();
```

Umwandlung der zufälligen Menge in eine balancierte Sequenz

```
for i from 1 to 2*n do
  if (member(i, menge)) then k[i] := 1;
  else k[i] := 2; fi; od;
sequenz := table();
phi(k,sequenz,1,2*n,1,2*n);
RETURN(eval(sequenz));
end:
```

```
phi := proc(k,sequenz,as,ns,ak,nk)
```

berechnet die Funktion $phi_{Laenge\ von\ k}$ *von der Sequenz k*
as, ns: Beginn und Ende des zu beschreibenden Teils von sequenz
ak, nk: Beginn und Ende des zu betrachtenden Teils von k

```
local x, zaehler, i,j;
```

x: irreduzible Teilsequenz von k
i: Länge von x

```
if ak>nk then RETURN();
else
```

Finden der irreduziblen Teilsequenz x:

```
  x:= table([(1)=k[ak]]);
  if (k[ak] =1) then zaehler :=1;
  else zaehler := -1; fi;
  i:= 1;
  while (zaehler <>0) do
```

```
    x[i+1] := k[ak+i];
    if (k[ak+i] =1) then zaehler := zaehler +1;
    else zaehler := zaehler -1; fi;
    i:= i+1; od;
  if (x[1]=1) then
```
x gültig
```
    for j from 1 to i do sequenz[as+j-1] :=x[j]; od;
    phi(k,sequenz,as+i,ns,ak+i,nk);
```
x in sequenz übernehmen und den Rest von k betrachten
```
  else
```
x nicht gültig
```
    stern(k,ak+1,ak+i-2);
```
t bilden*
```
    phi(k,sequenz,as+nk-(ak-1)-i+2,ns,ak+1,ak+i-2);
```
$\phi(t^*) = t^*$ *ans Ende von sequenz schreiben*
```
    phi(k,sequenz,as+1,as+nk-(ak-1)-i,ak+i,nk);
```
$\phi(y)$ *in sequenz schreiben*
```
    sequenz[as] := 1; sequenz[as+nk-(ak-1)-i+1] := 2; fi; fi;
end:
```

```
stern := proc(k,a,e)
```
berechnet $k[a..e]^*$
```
local i;
tstern :=[];
for i from a to e do
  if (k[i] =1) then k[i] := 2;
  else k[i] := 1; fi; od;
RETURN(tstern);
end:
```

A.2.2 Algorithmus von Arnold und Sleep

Statt einer Schleife mit der Bedingung $s > 0$ gibt es im Programm einige kleinere Änderungen, um Rundungsfehlern bei der Berechnung der Wahrscheinlichkeit vorzubeugen. Im Fall $r = 0$ ist die Wahrscheinlichkeit, als nächstes Zeichen ein ρ zu haben, gleich 0, d.h. das erste Zeichen ist in jedem Fall eine öffnende Klammer, im Fall $r = s$ ist sie 1, d.h. die restlichen Zeichen sind schließende Klammern. Es wird also in beiden Fällen keine Zufallsauswahl benötigt.

```
arnsleep := proc(n)
```
erzeugt eine zufällige Klammersequenz aus n Klammerpaaren mit dem Algorithmus von Arnold und Sleep
```
local s, r, P, sequenz, rno, i;
```

r: Anzahl noch zusätzlich benötigter schließender Klammern
s: Anzahl noch zu erzeugender Zeichen
P: Wahrscheinlichkeit für eine schließende Klammer
sequenz: die zu erzeugende Klammersequenz

```
sequenz:= table([(1)=1]); i :=2;
r:=1;
s:=2*n-1;
while r <> s do
  P:= r*(r+s+2)/(2*s*(r+1));
  rno := 10^(-12)*rand();
  if rno <= P then
```

nächstes Zeichen ist eine schließende Klammer

```
    sequenz[i] := 2;
    r:= r-1;
  else
```

nächstes Zeichen ist eine öffnende Klammer

```
    sequenz[i] := 1;
    r:= r+1; fi;
  s:= s-1; i := i+1; od;
while not(s =0) do
  sequenz[i] := 2;
  s:= s-1; i := i+1; od;
RETURN(eval(sequenz));
end:
```

A.3 Erzeugung von binären Bäumen

A.3.1 Algorithmus von Knott

```
knott := proc(n,Catalanzahlen)
```

berechnet einen zufälligen Baum mit n inneren Knoten, Catalanzahlen ist die Liste der Catalanzahlen von 0 bis n

```
local rn, wert, baum;
```

wert: zufällig bestimmter Rang

```
rn := rand(1..Catalanzahlen[n]);
wert := rn();
baum := table();
rang_1(wert,n,Catalanzahlen,0,1);
RETURN(baum_aus_perm(baum,0,1,n));
end:
```

```
rang_1 := proc(wert,n,Catalanzahlen,plus,a)
```

berechnet den Baum b zu rang(b) nach der Rangfunktion von Knott als Permutation dargestellt
plus ist die Konstante, die in rechten Teilbäumen hinzuaddiert werden muß, a ist der Anfang der in baum zu beschreibenden Teilpermutation

```
local j, i, irechts, ilinks,x;
```

j: Anzahl der Knoten im linken Teilbaum
ilinks, irechts: Rang des linken bzw. rechten Teilbaums
i : rang(b)

```
if n=0 then RETURN(); fi;
if n=1 then baum[a] := plus+1; RETURN(); fi;
j:=0;
i:=wert;
```

Bestimmung der Anzahl Knoten im linken Teilbaum:

```
while (i>G(j,n,Catalanzahlen)) do
  i:=i-G(j,n,Catalanzahlen);
  j:= j+1; od;
irechts := 1+((i-1) mod Catalanzahlen[n-j-1]);
ilinks := (i-irechts)/Catalanzahlen[n-j-1]+1;
```

der Baum setzt sich aus der Permutation für den linken Teilbaum und der entsprechend erhöhten Permutation des rechten Teilbaums zusammen:

```
baum[a]:= j+1+plus;
rang_1(ilinks,j,Catalanzahlen,baum,plus,a+1)),
rang_1(irechts,n-j-1,Catalanzahlen,baum,plus+j+1,a+j+1);
RETURN();
end:
```

```
G := proc(j,n,Catalanzahlen)
```

berechnet G(j,n) aus der Catalanzahl-Liste

```
RETURN(Catalanzahlen[j]*Catalanzahlen[n-j-1]);
end:
```

```
baum_aus_perm := proc(perm,vaternr,a,n)
```

berechnet den Baum, der durch die Permutation gegeben ist, in der üblichen Darstellung, vaternr ist die Nummer des Vaters des gerade betrachteten rechten Teilbaumes, anfangs =0, a ist der Beginn der in perm zu betrachtenden Teilpermutation, n bezeichnet das Ende der gerade betrachteten Teilpermutation

```
local links, rechts, nr, baum;
```

links, rechts: Permutationen des linken bzw. rechten Teilbaums
nr: Nummer der Wurzel= 1+ Anzahl innerer Knoten im linken Teilbaum

```
if (a>n) then RETURN([]); fi;
nr := perm[a]-vaternr;
links := baum_aus_perm(perm,vaternr,a+1,a+nr-1);
rechts:= baum_aus_perm(perm,vaternr+nr,a+nr,n);
```

```
baum:=[links, rechts];
RETURN(baum);
end:
```

A.3.2 Algorithmus von Ruskey und Hu

```
ranghu_1:=proc(rang,n)
```
berechnet den Baum mit n Blättern mit Rang rang nach der Rangfunktion von Ruskey und Hu
```
local r,m,l,j,tsum,a,i,probe,wert;
```
r: reduzierte Sequenz
m: kleinste Zahl, die die bisher erzeugte Sequenz gültig erweitert
l: Länge der reduzierten Sequenz
tsum: Summe über die T(n-j,m-l+i)
a: die erzeugte Sequenz
wert: noch nicht verarbeiteter Rang
```
wert := rang;
r:=table([(1)=0]);
m:=1;
l:=0;
a:= table();;
for j from 1 to n-1 do
```
bestimme minimales i mit Summe T(n-j,m-l+x) für x von 0 bis i+1 ist kleiner wert:
```
  i:=0;
  tsum := 0;
  probe :=0;
  if (wert <=tsum) then probe :=1; fi;
  while probe =0 do
   tsum := tsum + T(n-j,m-l+i);
   i:=i+1;
   if (wert <=tsum) then probe :=1; fi; od;
 wert := wert-tsum+T(n-j,m-l+i-1);
```
a entsprechend setzen:
```
 a[j]:= m+i-1;
```
r setzen und gegebenenfalls reduzieren:
```
 l:=l+1;
 r[l+1] := a[j];
 while (r[l]=r[l+1]) do
  l:=l-1;
  r[l+1] := r[l+1]-1; od;
 if (r[l+1]=l) then m:=l+1;
```

```
 else m:= r[l+1]; fi; od;
```

a[n] setzen:

```
a[n] := r[l+1];
RETURN(eval(a));
end:
```

```
T := proc(n,k)
```

berechnet T(n,k)

```
RETURN(k/(2*n-k)*(2*n-k)!/((n-k)!*n!));
end:
```

```
ruskeyhu := proc(n,Catalanzahlen)
```

erzeugt zufällig einen Baum mit Hilfe der Rangfunktion von Ruskey und Hu

```
local rn, wert;
rn := rand(1..Catalanzahlen[n]);
wert := rn();
RETURN(baum_aus_blattlevel(ranghu_1(wert,n+1)));
end:
```

```
baum_aus_blattlevel:=proc(level)
```

berechnet den Baum, der der Blattlevelsequenz entspricht

```
local baum;
baum := teilbaum(level,1,0)[1];
```

an erster Stelle von teilbaum steht der Baum

```
RETURN(baum);
end:
```

```
teilbaum := proc(level,a,ebene)
```

berechnet den Baum, der der Blattlevelsequenz entspricht, a ist der Anfang der zu betrachtenden Sequenz, ebene ist die Ebene im baum, auf der man sich während der Bearbeitung gerade befindet

Rückgabe: [Baum, Anfang der noch nicht verarbeiteten Sequenz]

```
local links, rechts,i;
```

links, rechts: linker bzw. rechter Teilbaum

```
if level[a] = ebene then
```

d.h. auf dieser Ebene gibt es ein Blatt

```
  RETURN([[],a+1]);
else
```

die Teilbäume eine Ebene tiefer bestimmen:

```
  links := teilbaum(level,a,ebene+1);
  rechts := teilbaum(level,links[2],ebene+1);
  RETURN([[links[1],rechts[1]],rechts[2]]); fi;
end:
```

A.3.3 Schrittweises Wachsen von Bäumen

```
baumwachsen := proc(n,L)
```
erzeugt einen zufälligen Baum durch schrittweises Wachsen
```
local x, j, w_keit, e, i, zufall;
```
x: Liste der Endblätter, die erweitert werden
e: Liste der Anzahlen von Endblättern
w_keit: Wahrscheinlichkeit, mit der ein Blatt ausgewählt wird
j: jeweils ausgewähltes Endblatt
```
zufall := rand()*10^(-12);
```
Festlegung des ersten zu erweiternden Blattes:
```
if zufall <= (L[3,n-2]/L[2,n-1]) then x:=table([(1)=0]);
else x:=table([(1)=1]); fi;
e:=table([(1)=2]);
for i from 2 to n-1 do
```
Bestimmung der Anzahl von Endblättern
```
  e[i] := e[i-1]+1-x[i-1];
```
zufällig ein Endblatt auswählen:
```
  zufall := rand()*10^(-12);
  w_keit := L[e[i]+1,n-1-i]/L[e[i],n-i];
  j:=0;
  while zufall > w_keit do
    j:=j+1;
    w_keit := w_keit + L[e[i]+1-j,n-1-i]/L[e[i],n-i]; od;
  x[i] := j; od;
RETURN(baum_aus_pfadh(pfadh_aus_endblaetter(x,n-1)));
end:
```

```
L_liste := proc(n)
```
liefert eine Tabelle der Werte L(m,j) für m von 0 bis n-1 und j von 0 bis n+1-m
```
local L, m, j;
L:= table();
for m from 0 to n+1 do
  L[m,0]:=1;
```
nach Eigenschaft 1 von L
```
  L[m,1] := m; od;
```
nach Eigenschaft 2 von L
```
for j from 2 to n+1 do
  L[1,j] := 1/(j+1)*(2*j)!/(j!*j!);
```
L(1,j) = Anzahl der Bäume mit j+1 Blättern
```
  L[2,j] := 1/(j+2)*(2*j+2)!/((j+1)!*(j+1)!); od;
```
nach Eigenschaft 3 von L
```
for m from 3 to n-1 do
```

```
  for j from 2 to n+1-m do
    L[m,j] := L[m-1,j+1]-L[m-2,j+1]; od; od;
```
nach Eigenschaft 4 von L
```
RETURN(eval(L));
end:
```

```
pfadh_aus_endblaetter := proc(sequenz,anz)
```
berechnet aus der Endblättersequenz eines Baumes mit anz Einträgen den entsprechenden Hilfspfad
```
local pfad, i, v;
```
v := Stelle im Pfad, die dem zu erweiternden Endblatt entspricht
```
pfad := [1,-1];
```
Anfangspfad, entspricht dem Baum mit einem inneren Knoten
```
v := 1;
for i from 1 to anz do
  v := v+sequenz[i];
  pfad := [teil(pfad,1,v),1,-1,teil(pfad,v+1,nops(pfad))];
```
das Erweitern eines Blattes im Baum entspricht dem Einfügen einer Spitze im Pfad
```
  v := v+1; od;
RETURN(pfad);
end:
```

```
baum_aus_pfadh:= proc(pfadh)
```
berechnet einen Baum aus dem Hilfsfad pfad
```
RETURN(b_a_ph(pfad,1)[1]);
```
in der Rückgabeliste von t_a_p steht der eigentliche Baum an erster Stelle
```
end:
```

```
b_a_ph:= proc(pfadh,i)
```
baut den zu Pfad gehörigen Baum mit Hilfe einer Tiefensuche auf die Rückgabe hat die Form [Baum, noch nicht verarbeiteter Restpfad]
```
local baum, links, rechts;
```
links, rechts: linker bzw. rechter Teilbaum mit verbliebenem Restpfad
```
if i>nops(pfadh) or pfadh[i]=-1 then RETURN([[],i]); fi;
```
pfad[1]=-1 bedeutet, daß in pfadh ein rechter Teilbaum beginnt
```
links:=b_a_ph(pfadh,i+1);
rechts := b_a_ph(pfadh,links[2]+1);
baum:= [[links[1],rechts[1]],rechts[2]];
```
der Rückgabebaum setzt sich aus den beiden Teilbäumen und dem noch verbliebenen Restpfad zusammen
```
RETURN(baum);
end:
```

A.4 Erzeugung von Dyck-Pfaden

```
dyck_pfade := proc(n,const)
erzeugt einen zufälligen Dyck-Pfad mit Hilfe einer Markov-Kette,
const ist der Faktor für die Bestimmung der Anzahl auszuführender Schritte
local pfad, p, r, T,i,rnp, rnr;
rnp, rnr: Zufallsgenerator für die Auswahl von p bzw. r
T: Anzahl auszuführender Schritte
pfad := table([(0)=0]);
Anfangspfad ist der Zickzack-Pfad von (0,0) nach (2n,0):
for i from 1 to n do
  pfad[2*i-1] := 1; pfadh[2*i] := 0; od;
rnr := rand(0..1);
rnp := rand{1..2*n-1);
T := evalf(ceil(const* n^3 * log[2](n);
Festlegung der Anzahl auszuführender Schritte
p := rnp();
r := rnr();
r=0 entspricht einem Aufwärtsflip, r=1 einem Abwärtsflip
Flip ausführen, falls dies möglich ist:
  if (r=0 and pfad[p-1]=pfad[p+1] and pfad[p-1]=pfad[p]+1) then
    pfad[p] := pfad[p]+2 fi;
  if (r=1 and pfad[p-1]=pfad[p+1] and pfad[p-1]=pfad[p]-1
      and pfad[p-1]>0) then
    pfad[p] := pfad[p]-2 fi; od;
RETURN(eval(pfad));
end:
```

A.5 Erzeugung von Triangulierungen

A.5.1 Gleichverteilung

```
flip := proc(t,nr)
flippt in der Tabellen-Triangulierung t die Diagonale mit Nummer nr
local p1, p2, p3, p4, d1, d2, d3, d4;
p1 := t[nr,1];
p2 := t[nr,2];
p1-p2 ist die Diagonale
p3 := t[nr,3];
p4 := t[nr,4];
das entstehende Viereck bei Entfernung der Diagonale ist p1-p4-p2-p3 im Uhr-
zeigersinn
```

```
d1 := t[nr,5];
```
Diagonale p1-p3
```
d2 := t[nr,6];
```
Diagonale p1-p4
```
d3 := t[nr,7];
```
Diagonale p2-p3
```
d4 := t[nr,8];
```
Diagonale p2-p4

der eigentliche Flip:
```
t[nr,1] := p3;
t[nr,2]:=p4;
t[nr,3]:=p2;
t[nr,4]:=p1;
t[nr,5]:= d3;
t[nr,6]:= d1;
t[nr,7]:= d4;
t[nr,8]:= d2;
```
die Werte für die Begrenzungsdiagonalen des Vierecks entsprechend verändern:
```
if d1 <> 0 then
```
d1=0 heißt, daß eine Außenkante erreicht ist, für sie muß nichts geändert werden
```
   if (t[d1,3] = p2) then
```
neuen 4. Punkt im Viereck von d1 setzen:
```
      t[d1,3] := p4;
      t[d1,5] := nr;
      t[d1,7] := d2;
 else
```
neuen 2. Punkt im Viereck von d1 setzen:
```
    t[d1,4] := p4;
    t[d1,6] := d2;
    t[d1,8] := nr; fi; fi;
```
genauso für die anderen Diagonalen:
```
if d2 <> 0 then
  if (t[d2,3] = p2) then
    t[d2,3] := p3;
    t[d2,5] := d1;
    t[d2,7] := nr;
else
    t[d2,4] := p3;
    t[d2,6] := nr;
    t[d2,8] := d1; fi; fi;
if d3 <> 0 then
if (t[d3,3] = p1) then
    t[d3,3] := p4;
```

```
    t[d3,5] := d4;
    t[d3,7] := nr;
else
    t[d3,4] := p4;
    t[d3,6] := nr;
    t[d3,8] := d4; fi; fi;
if d4 <> 0 then
  if (t[d4,3] = p1) then
    t[d4,3] := p3;
    t[d4,5] := nr;
    t[d4,7] := d3;
else
    t[d4,4] := p3;
    t[d4,6] := d3;
    t[d4,8] := nr; fi; fi;
RETURN(eval(t));
end:
```

```
anfangstriangulierung := proc(n)
```

erzeugt die Tabelle, die der Fächertriangulierung von Punkt 1 aus entspricht, mit Angabe der zu den Diagonalen gehörigen Vierecke

```
local t, i;
t := table();
for i from 1 to n-3 do
```

Diagonale von 1 zu den anderen Polygonecken:

```
  t[i,1] := i+2;
  t[i,2] := 1;
```

Die anderen beiden Viereckspunkte:

```
  t[i,3] := i+1;
  t[i,4] := i+3;
```

Die anderen Vierecksdiagonalen, zwei sind jeweils Außenkanten:

```
  t[i,5] := 0;
  t[i,6] := 0;
  t[i,7] := i-1;
  t[i,8] := (i+1) mod (n-2); od;
RETURN(eval(t));
end:
```

```
zufaellige_triangulierung := proc(n, const)
```

erzeugt eine zufällige Triangulierung mit Hilfe einer Markov-Kette,
const ist der Faktor für die Anzahl der auszuführenden Schritte

```
local t, nr, rn, i, T;
```

t: Tabellen-Triangulierung
nr: zum Flip ausgewählte Diagonale

T: Anzahl auszuführender Flips

```
t:= anfangstriangulierung(n+2);
rn := rand(1..n-1);
T :=  evalf(ceil(const * (n+2)^5* log[2](n+2)));
for i from 1 to T do
  nr := rn();
  t := flip(t,nr); od;
RETURN(verkuerzen(eval(t),n+2));
```

liefert die Triangulierung in der üblichen Form als Tabelle der Diagonalen

```
end:
```

```
verkuerzen := proc(t_lang,n)
```

berechnet aus der Flip-Triangulierung eine Tabelle der Diagonalen

```
local t, i;
t := table();
for i from 1 to n-3 do
  t[i,1]:=t_lang[i,1]; t[i,2] := t_lang[i,2]; od;
```

die Diagonalen sind jeweils die ersten beiden Tabelleneinträge

```
RETURN(eval(t));
end:
```

A.5.2 Triangulierung mit fester längster Diagonale

```
diag_triag := proc(n, k)
```

erzeugt eine zufällige Triangulierung, deren längste Diagonale die Länge k hat

```
local i, j, U, I, triag1, triag, S, C, rn;
```

U: zur Wahrscheinlichkeitsberechnung
triag1: Triangulierung während eines Zwischenschrittes
triag: Ausgabetriangulierung
S: Anzahl der möglichen Triangulierungen rechts von der längsten Diagonalen
C: Liste der Catalanzahlen

```
U := rand() * 10 ^(-12);
C:=catalanliste(n+2);
```

bestimme S:

```
S:=0;
if k<(n+2)/2 then
  for i from n+2-k to 2*k do S := S+C[i-k]*C[n+2-i]; od;
```

wähle i entsprechend den errechneten Wahrscheinlichkeiten:

```
  i:=n+2-k;
  j := C[i-k]*C[n+2-i];
  while j < U*S do
    i:=i+1;
```

```
    j := j+C[i-k]*C[n+2-i]; od;
```
genauso für k=(n+2)/2
```
else for i from n+3-k to 2*k-1 do
```
S := S+C[i-k-1]*C[n+1-i]; od;!
```
  i:=n+3-k;
  j := C[i-k-1]*C[n+1-i];
  while j < U*S do
    i:=i+1;
    j := j+C[i-k-1]*C[n+1-i]; od; fi;
```
die längste Diagonale einfügen:
```
triag1:=table([(1,1)=0,(1,2)=k]);
```
trianguliere das (k+1)-gon links dieser Diagonalen:
```
if k>=3 then
  triag := triangulierung_aus_baum(baum_aus_pfad
           (pfad_aus_klammern(atksack(k-1),k-1),k-1),k-1);
  for j from 1 to k-2 do
    triag1[j+1,1] :=triag[j,1]-1;
    triag1[j+1,2]:=triag[j,2]-1; od; fi;
```
trianguliere das Teilpolygon, das von i-k begrenzt wird:
```
if i-k>1 then triag1[k,1]:=k; triag1[k,2] := i; fi;
if i-k>=3 then
  triag:=triangulierung_aus_baum(baum_aus_pfad(pfad_aus_klammern
         (arnsleep(i-k-1),i-k-1),i-k-1),i-k-1);
  for j from 1 to i-k-2 do
    triag1[j+k,1] :=triag[j,1]+k-1;
    triag1[j+k,2]:=triag[j,2]+k-1; od; fi;
```
trianguliere das dritte Teilpolygon:
```
if n+2-i>1 then triag1[i-1,1] := i; triag1[i-1,2]:=0; fi;
if  i < n then
  triag := triangulierung_aus_baum(baum_aus_pfad(pfad_aus_klammern
           (arnsleep(n+1-i),n+1-i),n+1-i),n+1-i);
  for j from 1 to n-i do
    triag1[j+i-1,1 := (triag[j,1]+i-1)mod(n+2);
    triag1[j+i-1,2] := (triag[j,2]+i-1)mod(n+2)]]; od; fi;
```
rotiere die entstandene Traingulierung um ein zufällig gewähltes I:
```
rn :=rand(n+1);
I := rn();
for j from 1 to n-1 do
  triag[j,1]:=(triag1[j,1]+I)mod(n+2)+1;
  triag1[j,2]:=(triag1[j,2]+I)mod(n+2)+1]]; od:
RETURN(eval(triag1));
end:
```

A.5.3 Triangulierung mit fester Anzahl von Ohren

```
ohr_triag := proc(n,k)
```
erzeugt eine zufällige Traingulierung mit k Ohren
```
local t1, i, S, t, I, variante, S_quer, j, h, t2, letzte;
```
t1, t2: Hilfstriangulierungen für Zwischenschritte
t: Ausgabetriangulierung
S: zufällige Menge als Nummern der schon vorhandenen Diagonalen
I: Faktor, um den rotiert werden soll
variante: Nummer der Variante, nach der neue Diagonalen eingefügt werden
S_quer: Komplement zu S
j: zu erweiternde Diagonale
letzte: Diagonale 1-(n+1)
trianguliere das k-gon zufällig und füge die Außenkanten des k-gons ein:
```
if k>3 then
  t1:=zufaellige_triangulierung(k-2);
  t1 :=[seq([i[1]-1,i[2]-1],i=t1)];
  t1 :=[seq([2*i[1]+1,2*i[2]+1],i=t1)];
  for i from 1 to k do t1 := [op(t1),[2*i-1,(2*i+1)mod(2* k)]]; od;
else if k=2 then t1 :=[[1,3]];
     else t1 := [[1,3],[3,5],[1,5]]; fi; fi;
```
wähle eine zufällige Menge S als Nummern für die jetzt vorhandenen Diagonalen:
```
S := zufaellig(n-2,2*k-4);
S:=sort([op(S)]);
S_quer := [op(({$1..n-2} minus {op(S)}))];
S_quer := sort(S_quer);
```
die Diagonalen in t1 werden gemäß der erwähnten Ordnung sortiert, um die Nummern in S zuordnen zu können:
```
t1 :=map(sort,t1);
letzte := t1[nops(t1)];
t1 := [op(sort([teil(t1,1,nops(t1)-1)],diagordnung)),letzte];
```
in t2 werden die Diagonalen aus t1 mit Nummern aus S durchnummeriert:
```
t2:=[];
for i from 1 to nops(S) do t2:=[op(t2),[S[i],t1[i][1],t1[i][2]]]; od;
t2 := [op(t2),[n-1,t1[nops(t1)][1],t1[nops(t1)][2]]];
```
finde eine zu erweiternde Diagonale:
```
j:=1;
for i from 1 to n-2-(2*k-4) do
  while S_quer[i]>t2[j][1] do j:=j+1; od;
  variante := rand(1..2)();
```
Erweiterungsvariante auswählen und neue Diagonale einfügen:
```
  if variante =1 then
    t2:= erhoehen1(t2,t2[j][2],t2[j][3]);
```

```
    t2 := [teil(t2,1,j-1),[S_quer[i],t2[j][2],t2[j][3]],
           [t2[j][1],t2[j][2]+1,t2[j][3]],teil(t2,j+1,nops(t2))];
    j:=j+1;
  else
    t2 := erhoehen2(t2,t2[j][2],t2[j][3]);
    t2 := [teil(t2,1,j-1),[S_quer[i],t2[j][2],t2[j][3]],
           [t2[j][1],t2[j][2],t2[j][3]+1],teil(t2,j+1,nops(t2))];
  fi; od;
```
rotiere um ein zufällig gewähltes I:
```
I:=rand(n+1)();
t:=[seq([(h[2]+I)mod(n+2)+1,(h[3]+I)mod(n+2)+1],h=t2)];
RETURN(t);
end:
```

```
diagordnung:=proc(d1,d2)
```
entscheidet, ob nach der erwähnten Diagonalenordnung d1 ¡ d2 ist
```
RETURN((d1[1]=d2[1] and d1[2]<d2[2]) or (d1[1]<d2[1]));
end:
```

```
erhoehen1:=proc(t,x,y)
```
ändert die Diagonalen x-y bei Einfügen einer neuen Diagonalen nach Variante 1
```
local t1, i;
t1:=[];
for i from 1 to nops(t) do
  if t[i][2]> x or (t[i][2]=x and t[i][3]<y) then
    t1 := [op(t1),[t[i][1],t[i][2]+1,t[i][3]+1]];
 else if t[i][3]>x then
           t1 := [op(t1),[t[i][1],t[i][2],t[i][3]+1]];
  else t1 :=[op(t1),t[i]]; fi; fi; od;
RETURN(t1);
end:
```

```
erhoehen2:=proc(t,x,y)
```
ändert die Diagonalen bei Einfügen einer neuen Diagonalen x-y nach Variante 1
```
local t1, i;
t1:=[];
for i from 1 to nops(t) do
  if t[i][2]>= y then
    t1 := [op(t1),[t[i][1],t[i][2]+1,t[i][3]+1]];
  else if t[i][3] > y or (t[i][3]=y and t[i][2]<x) then
          t1 :=[op(t1),[t[i][1],t[i][2],t[i][3]+1]];
       else t1 := [op(t1),t[i]]; fi; fi; od;
RETURN(t1);
end:
```

Literatur

[1] D.J. Aldous, *Random Walks on Finite Groups and Rapidly Mixing Markov Chains*, Lecture Notes in Mathematics 986, Springer Verlag

[2] D.J. Aldous, *Some Inequalities for Reversible Markov Chains*, Journal of the London Mathematical Society (2) 25, S. 564-576

[3] D.B. Arnold, M.R. Sleep, *Uniform Random Generation of Balanced Parenthesis Strings*, ACM Transaction Programming Languages Systems 2 (1980), S. 122-128

[4] M.D. Atkinson, J.R. Sack, *Generating Binary Trees at Random*, Information Processing Letters 41 (1992) S. 21-23

[5] L. Devroye, P. Flajolet, F. Hurtado, M. Noy, W. Steiger, *Properties of Random Triangulations and Trees*, Discrete and Computational Geometry 22 (1999), S. 105-117

[6] P. Diaconis, L. Saloff-Coste, *Comparison Theorems for Reversible Markov Chains*, The Annals of Applied Probability 3 (1993) S.696-730

[7] P. Diaconis, D. Stroock, *Geometric Bounds for Eigenvalues of Markov Chains.* The Annals of Applied Probability 1 (1991), S. 36-61

[8] R. Horn, C. Johnson, *Matrix Analysis*, Cambridge University Press, 1985

[9] B. Johnson, *Generating Binary Trees with Uniform Probability*, BIT 31 (1991) S. 15-31

[10] G.D. Knott, *A Numbering System for Binary Trees*, Communications of the ACM 20 (1977), S. 113-115

[11] M. Luby, D. Randall, A. Sinclair, *Markov Chain Algorithms for Planar Lattice Structures*, Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (1995)

[12] L. McShine, P. Tetali, *On the Mixing Time of the Triangulation Walk and other Catalan Structures*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 43 (1999), Herausgeber: P.Pardalos et al.

[13] M. Molloy, B. Reed, W. Steiger, *On the Mixing Rate of the Triangulation Walk*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 43 (1999), Herausgeber: P. Pardalos et al.

[14] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995

[15] J.G. Propp, D.B. Wilson, *Exact Sampling with Coupled Markov Chains and Applications to Statistical Mechanics*, Extended Abstract (1996)

[16] D. Randall, P. Tetali, *Analyzing Glauber Dynamics by Comparison of Markov Chains*, Lecture Notes in Computer Science 1380 (1998)

[17] E.M. Reingold, J. Nievergelt, N.Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, New Jersey, 1977

[18] F. Ruskey, T.C. Hu, *Generating of Binary Trees Lexicographically*, SIAM Journal of Computation 6 (1977), S. 745-758

[19] R. Sedgewick, *Algorithmen*, Addison-Wesley Verlag, 1992

[20] A. Sinclair, *Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow*, Combinatorics, Probability and Computing 1 (1992), S. 351-370

[21] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov Chain Approach*, Birkhäuser, Boston 1992

[22] D. Sleator, R. Tarjan, W. Thurston, *Rotation Distance, Triangulations and Hyperbolic Geometry*, Journal of the American Mathematical Society 1 (1988), S. 647-681

[23] R.P. Stanley, *Enumerative Combinatorics, vol 2*, Cambridge University Press, 1998

[24] D.B. Wilson, Manuscript