Algorithmen der Bioinformatik

Prof. Dr. Sven Rahmann
Lehrstuhl XI, Fakultät für Informatik, TU Dortmund

INF-MSc-606, Sommersemester 2013
INF-MSc-606, Sommersemester 2016
Entwurf vom 7. Dezember 2017

Inhaltsverzeichnis

Vorbemerkungen

Dieses Skript enthält Material der Vorlesung „Algorithmische Bioinformatik“ (als Modul INF-MSc-606), die ich an der TU Dortmund seit dem Sommersemester 2013 gehalten habe. Gegenüber früheren Versionen der Vorlesung ist das Material nun moderner und stärker stochastisch ausgerichtet. Zentrales Thema sind stochastische Modelle für molekulare Daten, wie sie in der Bioinformatik typischerweise anfallen.

Vorausgesetzt werden Grundlagenkenntnisse über Sequenzen, wie sie auch in der Vorlesung „Algorithmen auf Sequenzen“ vermittelt werden. Man sollte also wissen, was sich hinter Begriffen wie Alphabet, Präfix, Suffix, Teilstring, Teilsequenz, etc. verbirgt. Die genaue Kenntnis der Algorithmen aus „Algorithmen auf Sequenzen“ ist nicht notwendig, aber hilfreich.

Das Skript befindet sich zur Zeit noch in der Entwurfsphase; es ist somit wahrscheinlich, dass leider noch einige Fehler darin enthalten sind.

Dortmund, April 2013 Sven Rahmann

KAPITEL 1

Simulation von Zufallssequenzen

Die Analyse von Sequenzen (insbes. DNA, RNA, Proteine, aber auch abstraktere Sequenzen wie Abfolgen von positiven und negativen Ladungen in einem Polymer-Molekül) nimmt in der Bioinformatik eine herausragend wichtige Stellung ein. Häufig stellt sich zu einem erhaltenen Ergebnis die Frage, ob es denn nun *statistisch signifikant* ist, d.h., ob sich das Ergebnis erheblich von dem unterscheidet, was man bei einer zufälligen Sequenz erwarten würde. Nur dann kann man offenbar von einer interessanten Entdeckung ausgehen.

Als Beispiel betrachten wir ein Stück DNA der Länge 1000 und finden heraus, dass die Nukletodie A und T darin jeweils 270 mal vorkommen und C und G jeweils 230 mal vorkommen. Weicht die Verteilung von der Gleichverteilung signifikant ab? Man kann das in diesem Fall ausrechnen (Stichwort: Binomialverteilung); häufig sind die Fragen und Analysen jedoch wesentlich komplizierter! In dem Fall hilft man sich dann mit der Simulation und Analyse zufälliger Sequenzen und vergleicht die Ergebnisse zwischen echten und simulierten Sequenzen.

Also stellt sich ganz grundlegend die Frage, wie man zufällige Sequenzen simulieren kann. Das ist im Grunde sehr einfach, erfordert aber dennoch ein paar Betrachtungen zur Effizienz.

1.1 Grundproblem

Gegeben ist ein endliches Alphabet Σ aus M Buchstaben, o.B.d.A. $\Sigma := \{\, 0, \ldots, M-1 \,\}$. Gegeben ist ferner eine Wahrscheinlichkeitsverteilung auf den Buchstaben $\pi = (\pi_0, \ldots, \pi_{M-1})$, so dass $\pi_i \geq 0$ für alle i und $\sum_i \pi_i = 1$. Aufgabe ist, ein zufälliges Element aus Σ mit den durch π gegebenen Wahrscheinlichkeiten auszuwählen.

1.2 Einfache Methoden

Ist π die Gleichverteilung, also $\pi_i = 1/M$ für alle i, dann ist das Ziehen eines zufälligen Elements kein Problem, da die Standardbibliothek fast jeder Programmiersprache eine Funktion bietet, eine zufällige ganze Zahl gleichverteilt aus $\{0, \ldots, M-1\}$ zu ziehen. In Python geschieht dies mit `random.randrange(M)`.

Ist jedoch π eine beliebige Wahrscheinlichkeitsverteilung, ist etwas mehr Aufwand erforderlich. Dazu stellen wir uns das Einheitsintervall $[0, 1]$ eingeteilt in M Intervalle der Längen π_i vor: Das Intervall $[0, \pi_0[$ gehört zum Objekt (Buchstaben) 0, das Intervall $[\pi_0, \pi_0 + \pi_1[$ zum Objekt 1, usw. Allgemein sei $\Pi_i := \sum_{j=0}^{i} \pi_j$; dann gehört das Intervall $[\Pi_{i-1}, \Pi_i[$ zum Objekt i. Wir ziehen eine in $[0, 1[$ gleichverteilte Zufallszahl U (z.B. durch `random.random()` in Python) und suchen das zugehörige Intervall. Dazu gibt es mehrere Möglichkeiten.

Lineare Suche. Beginnend mit $i = 0$ ziehen wir von U die Länge des aktuellen Intervalls π_i ab, solange U positiv bleibt und erhöhen i um 1. Wird U erstmals negativ, geben wir den aktuellen Wert von i zurück. Dies entspricht einer linearen Suche von links nach rechts durch die Intervalle. Damit diese möglichst effizient verläuft, ist es zweckmäßig, die Wahrscheinlichkeiten der Größe nach fallend zu sortieren (und die Objektnamen entsprechend zu permutieren). Die worst-case Laufzeit bleibt aber $O(M)$. Für kleine Alphabete ist die lineare Suche ohne Permutation aufgrund ihrer Einfachheit die beste Wahl.

Binäre Suche. Durch Vorberechnen der Intervallgrenzen Π_i lässt sich das zu U gehörende Intervall mittels binärer Suche in $O(\log M)$ Zeit finden. Vorteil ist offensichtlich die logarithmische Zeit (in M); Nachteil ist, dass zusätzlicher Speicher für die Vorberechnung der Intervallgrenzen benötigt wird. Der Overhead für die binäre Suche macht sich relativ schnell bezahlt, so dass dies für mittelgroße Alphabete die beste Wahl ist.

Wenn nun das Alphabet sehr groß ist und man sehr viele zufällige Zeichen generieren will, kann auch die Zeit $O(\log M)$ zu lang sein. Es stellt sich die Frage, ob man ein Zeichen auch in konstanter Zeit generieren kann. Dies ist in der Tat möglich, und zwar mit der Alias-Methode, die im nächsten Abschnitt vorgestellt wird. Es sei nochmals gesagt, dass sich dies nur bei großem M und vielen zu generierenden Zeichen lohnt.

1.3 Alias-Methode

Ist das Alphabet sehr groß und werden viele Ziehungen benötigt, dann gibt es die Möglichkeit, nach einer linearen Vorverarbeitung, die linearen zusätzlichen Platz benötigt, jede Ziehung in konstanter Zeit zu erhalten. Dabei werden allerdings zwei gleichverteilte Zufallszahlen in $[0, 1[$ benötigt.

Die Idee ist, das Intervall $[0, M[$ zunächst in M Hauptintervalle konstanter Länge 1 zu unterteilen und zuerst ein Hauptintervall gleichverteilt auszuwählen. Jedes Hauptintervall wird nun (auf geschickte Weise) in maximal zwei Unterintervalle geteilt. Dabei gehört das eine Unterintervall von Hauptintervall $[i, i+1[$ zu Objekt i mit einem festzulegenden Anteil $q_i \in [0, 1]$, und der Rest (Anteil $1 - q_i$) zu einem anderen geeignet auszuwählenden Objekt

```
1  def aliastable(p):
2      """Erzeugt zur Verteilung p=[p[0],...,p[M-1]],
3      die zunaechst normalisiert wird,
4      die Nachschlagetabellen fuer die Alias-Methode aus
5      Alias-Labeln a[0:M] und Anteilen q[0:M], so dass:
6      q[i]+summe[ 1-q[j] : j s.d. a[j]=i ] = M*p[i]  f.a. i
7      """
8      p = tuple(p)
9      M = len(p)
10     p = normalized(p, target=M, positive=True)  # Summe jetzt M
11     light = [(i,prob) for (i,prob) in enumerate(p) if prob<=1]
12     heavy = [(i,prob) for (i,prob) in enumerate(p) if prob> 1]
13     a, q = [None]*M, [None]*M
14     while len(heavy)>0:  # Leichte gibt es immer
15         # Nimm irgendein leichtes Element i, dazu ein schweres j:
16         # (i mit q[i]=pi) kommt zusammen mit (a[i]=j mit 1-pi)
17         i, pi = light.pop()
18         j, pj = heavy.pop()
19         a[i], q[i] = j, pi
20         # Packe aktualisiertes j in richtige Liste
21         pj -= (1-pi)
22         Lj = light if pj<=1 else heavy
23         Lj.append((j,pj))
24     # Jetzt gibt es nur noch leichte Elemente mit Gewicht 1.0
25     for (i,prob) in light: a[i], q[i] = i, 1.0
26     return a, q
```

Abbildung 1.1: Python-Code zur Erstellung der Aliastabelle. Die in Zeile 10 aufgerufene Funktion `normalized` stellt sicher, dass alle Einträge der übergebenen Verteilung positiv sind (`positive=True`) und skaliert sie so, dass die Summe danach M ist (`target=M`). Sie ist leicht selbst zu implementieren.

$a_i \in \{0,\dots,M-1\}\setminus\{i\}$ („Alias“). Die Einteilung muss so vorgenommen werden, dass der Gesamtanteil für jedes Objekt i proportional zu π_i ist, also

$$\pi_i = \frac{q_i + \sum_{j:\,a_j=i}(1-q_j)}{M}.$$

Hat man passende q_i und a_i gefunden, ist die Ziehung einfach: Ziehe i gleichverteilt aus $\{0,\dots,M-1\}$ und dann u aus $[0,1[$. Ist $u<q_i$, gib i aus, sonst a_i.

Die wesentliche Frage dabei ist, ob zu gegebenem π stets passende Anteile q und Aliasse a existieren, so dass diese effizient vorberechnet und in zwei Tabellen gespeichert werden können. Der folgende Satz bejaht diese Frage.

1.1 Satz (Alias-Methode)**.** *Zu jeder Verteilung $\pi=(\pi_0,\dots,\pi_{M-1})$ auf $\{0,\dots,M-1\}$ gibt es Aliasse (a_i), sowie Anteile (q_i), $i=0,\dots,M-1$, so dass*

$$\frac{q_i + \sum_{j:\,a_j=i}(1-q_j)}{M} = \pi_i \quad \textit{für alle } i=0,\dots,M-1$$

gilt.

Beweis. Wir teilen die M Objekte i in *leichte* und *schwere* Objekte ein, je nachdem ob $\pi_i \leq 1/M$ oder $\pi_i > 1/M$. Es gibt mindestens ein leichtes Objekt (gäbe es nur schwere, wäre die Gesamtwahrscheinlichkeit größer als 1). Wenn es nur leichte Objekte gibt, haben alle die Wahrscheinlichkeit $1/M$, und wir haben den Trivialfall der Gleichverteilung. Wir werden im folgenden in jedem Schritt, solange es mindestens ein schweres Objekt gibt, ein beliebiges leichtes Objekt i mit einem beliebigen schweren Objekt j zusammen in das Hauptintervall i „packen“. Dabei wird Objekt i komplett abgearbeitet und danach nicht mehr betrachtet. Objekt j kann dabei zu einem leichten Objekt werden (aber nicht ganz verschwinden), oder schwer bleiben; in jedem Fall wird π_j entsprechend angepasst. Offensichtlich muss dabei $q_i = M \cdot \pi_i$ und $a_i = j$ gewählt werden. Da ein leichtes Objekt nach Bearbeitung verschwindet, und ein schweres Objekt nicht direkt verschwinden kann, ist jedes Objekt genau einmal leicht; jedes Hauptintervall bekommt also genau einmal ein q_i und a_i zugewiesen. □

Python-Code, der diesen konstruktiven Beweis implementiert, ist in Abbildung 1.1 dargestellt.

KAPITEL 2

Textmodelle

In diesem Kapitel stellen wir verschiedene Textmodelle unterschiedlicher Komplexität vor. Ein Textmodell beschreibt, nach welchen Regeln eine Zufallssequenz (auch Zufallstext) zu erzeugen ist. Allen hier vorgestellten Modellen ist gemeinsam, dass sie ein „endliches Gedächtnis“ haben, d.h., einen Zustand aus einer endlichen Menge verwalten. (Andere Textmodelle wären auf einem Computer mit endlichem Speicher auch nicht implementierbar.)

2.1 Grundlegende Definitionen

Es sei wie immer Σ ein endliches Alphabet. Wir unterscheiden grundlegend zwischen zwei Arten von Textmodellen:

1. Modelle, bei denen die Textlänge n fest vorgegeben ist; dabei wird häufig eine ganze Modellfamilie, also ein Modell für jedes $n \in \mathbb{N}_0$ angegeben;
2. Modelle, bei denen die Textlänge als Zufallsvariable mitmodelliert wird.

2.1 Definition (Textmodell mit fester Länge)**.** Ein Textmodell $\mathbb{P}_n$ ist ein Wahrscheinlichkeitsmaß, das für eine feste Länge n jedem String $s \in \Sigma^n$ eine Wahrscheinlichkeit $\mathbb{P}_n(s)$ zuweist, so dass $\sum_{s\in\Sigma^n} \mathbb{P}_n(s) = 1$.

2.2 Definition (Textmodell mit zufälliger Länge)**.** Ein Textmodell $\mathbb{P}$ ist ein Wahrscheinlichkeitsmaß, das jedem endlichen String $s \in \Sigma^*$ eine Wahrscheinlichkeit $\mathbb{P}(s)$ zuweist, so dass $\sum_{s\in\Sigma^*} \mathbb{P}(s) = 1$.

Wir betrachten in diesem Abschnitt nur Modelle, bei denen die Länge n ein gegebener Parameter ist. Zur Vereinfachung schreiben wir auch $\mathbb{P}$ statt $\mathbb{P}_n$, wenn n fest gewählt ist.

2.3 Definition (Zufallstext)**.** Ein Zufallstext der Länge n zum Textmodell $\mathbb{P}_n$ ist ein stochastischer Prozess $S = S_0, \ldots, S_{n-1}$ mit Indexmenge $\{\, 0, \ldots, n-1 \,\}$ und endlicher Wertemenge Σ (Alphabet), so dass $\mathbb{P}_n(S = s) = \mathbb{P}_n(s)$.

Bei einem Textmodell $\mathbb{P}_n$ sind drei Aufgaben von Interesse:

1. Zu gegebenem Text s und Modell $\mathbb{P}_n$ die Berechnung seiner Wahrscheinlichkeit $\mathbb{P}_n(s)$ im Modell.
2. Zu gegebenem Modell $\mathbb{P}_n$ die Simulation eines Zufallstextes der Länge n gemäß $\mathbb{P}_n$, d.h. die Ausgabe der Simulation soll Text $s \in \Sigma^n$ mit Wahrscheinlichkeit $\mathbb{P}_n(s)$ sein.
3. Zu gegebenem Text s die Schätzung der Modellparameter, so dass die Wahrscheinlichkeit $\mathbb{P}_n(s)$ maximal unter allen zugelassenen Modellen wird.

Wir stellen in den nächsten Abschnitten konkrete Textmodelle vor. Bei jedem Textmodell geben wir an, wie man einen Text simuliert und wie man die Wahrscheinlichkeit eines gegebenen Textes unter dem Modell berechnen kann. Bei der praktischen Anwendung werden Wahrscheinlichkeiten schnell sehr klein; möglicherweise so klein, dass sie im Rechner nicht mehr mit double-Genauigkeit darstellbar sind. Deshalb rechnet man in der Praxis oft mit logarithmierten Wahrscheinlichkeiten, wobei sich die Multiplikation zu einer Summation vereinfacht. Auf diese Weise können sehr kleine Wahrscheinlichkeiten hinreichend genau dargestellt werden. Details hierzu finden sich in Abschnitt 2.7.

In Abschnitt 2.8 gehen wir für einige Modellklassen auf das Thema Parameterschätzung ein, das in den folgenden Kapiteln vertieft wird.

2.2 Das i.i.d.-Modell (M0) und die Gleichverteilung (M00)

In diesem Modell wird jedes Zeichen in $s = s_1 \ldots s_n$ unabhängig von den anderen und identisch verteilt erzeugt (i.i.d.: independently and identically distributed, auch Markov-Modell nullter Ordnung, M0, genannt, s.u.). Dazu müssen wir angeben, mit welcher Wahrscheinlichkeit p_a jeweils Zeichen $a \in \Sigma$ gewählt wird. Die Modellparameter sind also ein Wahrscheinlichkeitsvektor $p = (p_a)_{a\in\Sigma}$, und es ist

$$\mathbb{P}(s) = \prod_{i=1}^{n} p_{s_i} = \prod_{a\in\Sigma} p_a^{n_a},$$

wobei $n_a := |\{i : s_i = a\}|$ die Anzahl aller Vorkommen a in s ist.

Setzt man zusätzlich voraus, dass auf Σ die Gleichverteilung vorliegt, also $p_a \equiv 1/|\Sigma|$ gilt, dann ist $\mathbb{P}(s) = 1/|\Sigma|^n$ für alle s der Länge n. Dies wird auch als i.i.d.-Modell mit Gleichverteilung oder als M00-Modell bezeichnet.

2.4 Beispiel (M0-Modell)**.** Wir betrachten das Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ und die Verteilung $p = (0.5, 0.3, 0.2)$. Das Auftreten des String $s = \mathtt{abacab}$ hat somit die Wahrscheinlichkeit

$$\mathbb{P}_n(s) = p_a \cdot p_b \cdot p_a \cdot p_c \cdot p_a \cdot p_b = p_a^3 \cdot p_b^2 \cdot p_c^1 = 0.00225\,.$$

$$\begin{aligned}\log \mathbb{P}_n(s) &= \log p_a + \log p_b + \log p_a + \log p_c + \log p_a + \log p_b \\ &= 3 \cdot \log p_a + 2 \cdot \log p_b + 1 \cdot \log p_c\,.\end{aligned}$$

♡

2.3 Das Markov-Modell (M1)

Hierbei bilden die Zeichen in s eine homogene Markovkette erster Ordnung, d.h. die Wahrscheinlichkeiten für die Wahl des $(i+1)$-ten Zeichens hängen vom i-ten Zeichen ab (nicht aber von der weiteren Vergangenheit, und auch nicht von der Position i). Anzugeben ist also eine Wahrscheinlichkeitsmatrix $P = (p_{ab})_{a,b\in\Sigma}$ mit Zeilensummen $\sum_{b\in\Sigma} p_{ab} = 1$ für alle $a \in \Sigma$. Das erste Zeichen wird nach gesondert anzugebender Verteilung p^0 ausgewählt. Damit ergibt sich als Wahrscheinlichkeit, dass genau $s = s_1 \dots s_n$ erzeugt wird,

$$\mathbb{P}(s) = p^0_{s_1} \cdot \prod_{i=2}^{n} p_{s_{i-1},s_i}.$$

2.5 Beispiel (M1-Modell)**.** Sei wieder $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ und $s = \mathtt{abacab}$. Sei ferner ein M1-Modell gegeben durch die Startverteilung $p^0 = (0.5, 0.3, 0.2)$ und die Übergangsmatrix

$$P = \begin{pmatrix} 0.1 & 0.8 & 0.1 \\ 0.7 & 0.1 & 0.2 \\ 0.7 & 0.2 & 0.1 \end{pmatrix}$$

Die Wahrscheinlichkeit das für Auftreten von s berechnet sich nun durch

$$\begin{aligned}\mathbb{P}(s) &= p^0_a \cdot P_{a,b} \cdot P_{b,a} \cdot P_{a,c} \cdot P_{c,a} \cdot P_{a,b} \\ &= 0.5 \cdot 0.8 \cdot 0.7 \cdot 0.1 \cdot 0.7 \cdot 0.8 \\ &= 0.01568\,.\end{aligned}$$

♡

Unter der Voraussetzung $p_{ab} > 0$ für alle a, b gibt es eine eindeutige stationäre Verteilung $\pi = (\pi_a)_{a\in\Sigma}$, die invariant gegenüber der Übergangsmatrix ist, für die also (als Zeilenvektor) $\pi \cdot P = \pi$ gilt. Sie berechnet sich als Lösung des Gleichungssystems

$$\pi \cdot (P - \mathrm{Id}_{|\Sigma|} \,|\, e) = (0 \dots 0 \,|\, 1);$$

dabei ist e der Vektor aus Einsen. Eine der ersten $|\Sigma|$ Spalten ist überflüssig, da die ersten $|\Sigma|$ Spalten linear abhängig sind.

Vertrauter sieht das Gleichungssystem aus, wenn man π nicht als Zeilen-, sondern als Spaltenvektor auffasst (und dafür $x = \pi^T$ schreibt). Dann liest sich das zu lösende System als

$$\left(\begin{array}{c} P^T - \mathrm{Id}_{|\Sigma|} \\ \hline 1 \dots 1 \end{array}\right) \cdot x = \left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline 1 \end{array}\right),$$

wobei eine beliebige der ersten $|\Sigma|$ Zeilen gestrichen werden kann, da diese Zeilen linear abhängig sind. Zu beachten ist, dass man hier die transponierte Übergangsmatrix P^T verwenden muss.

2.4 Das Markov-Modell k-ter Ordnung (Mk)

Statt nur einem Zeichen kann man auch allgemeiner $k \geq 1$ Zeichen zurück schauen, um die Wahrscheinlichkeiten für das aktuelle Zeichen festzulegen. Die letzten k Zeichen bezeichnet man dann auch als den aktuellen *Sequenzkontext*. Man erhält das Markov-Modell k-ter Ordnung oder Mk-Modell. Man muss für die Sequenzanfänge ebenfalls Sequenzkontexte von weniger als k Zeichen berücksichtigen. Zweckmäßigerweise schreibt man die Übergangswahrscheinlichkeiten $p_{y,b}$ mit $y \in \bigcup_{k'=0}^{k} \Sigma^{k'}$ und $b \in \Sigma$ in eine $\frac{|\Sigma|^{k+1}-1}{|\Sigma|-1} \times |\Sigma|$-Matrix, aus der man für jeden gegebenen Sequenzkontext der Länge k (und für Anfangsstücke kürzerer Länge) die Wahrscheinlichkeiten für das nächste Zeichen ablesen kann. Die Anzahl der hierfür festzulegenden Parameter erhöht sich exponentiell mit k. Da man diese Parameter im Normalfall aus vorhandenen Daten schätzen muss, verbieten sich zu hohe Werte von k.

Wenn man wieder voraussetzt, dass das Modell stationär ist, kommt man mit einer $|\Sigma|^k \times |\Sigma|$-Matrix aus, da sich die entsprechenden Wahrscheinlichkeiten für kürzere Kontexte berechnen lassen.

2.5 Markov-Modelle variabler Ordnung

Ein Kompromiss, um weniger Parameter zur Modellbeschreibung zu benötigen, aber dennoch flexibel zu sein, ist, die Kontextlänge abhängig vom Kontext variabel zu halten.

Beispiel: Wir betrachten das Alphabet $\{a, b, c\}$. Ist das letzte Zeichen a oder b, hängen die Wahrscheinlichkeiten für das aktuelle Zeichen nur davon ab. Ist es aber c, betrachten wir die Kontexte $ac, bc, cc, \varepsilon c$ differenziert. Die entsprechenden Wahrscheinlichkeiten lassen sich in einem Trie speichern, so dass man sie effizient findet, wenn man den aktuellen Kontext von rechts nach links liest.

2.6 Beispiel (Anwendung eines Markov-Modells variabler Ordnung, aus Schulz et al. (2008))**.** Sei $s =$ `gattaca`. Wir möchten nun die Produktionswahrscheinlichkeit von s unter dem durch Abbildung 2.1 gegebenen Modell bestimmen. Dabei verwenden wir immer den maximal möglichen Kontext. Dabei bezeichnet ε den leeren Kontext.

$$\begin{aligned}\mathbb{P}(s) &= p_{\varepsilon,g} \cdot p_{g,a} \cdot p_{a,t} \cdot p_{gat,t} \cdot p_{tt,a} \cdot p_{a,c} \cdot p_{\varepsilon,a} \\ &= 0.17 \cdot 0.88 \cdot 0.91 \cdot 0.46 \cdot 0.04 \cdot 0.03 \cdot 0.33 \\ &\approx 2.48 \cdot 10^{-5}\end{aligned}$$

♡

2.6 Allgemeine Textmodelle mit endlichem Gedächtnis

Alle genannten Textmodelle lassen sich wie folgt verallgemeinern: Das Modell ist zu jeder Zeit in einem bestimmten *Zustand* oder *Kontext* aus einer endlichen Menge C, zu Beginn im Kontext $c_0 \in C$. Anhand einer kontextspezifischen Verteilung wird ein zu emittierendes Symbol $\sigma \in \Sigma$ und ein neuer Kontext $c \in C$ gewählt. Es gibt also eine Funktion $\phi : C \times \Sigma \times C \to [0, 1]$, die jedem Tripel (c, σ, c') aus aktuellem Kontext, nächstem Symbol

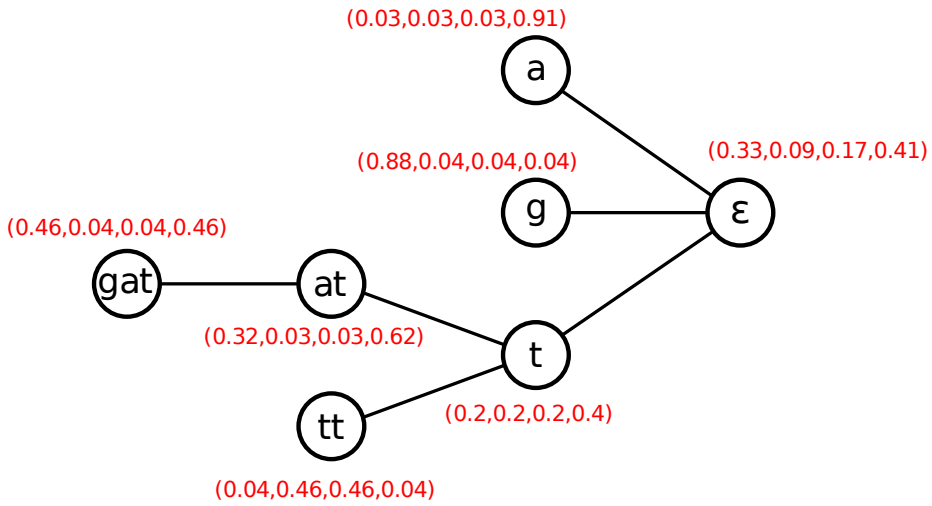


Abbildung 2.1: Schematische Darstellung eines Markov-Modells variabler Ordnung. In jedem Knoten steht der Kontext, der durch diesen Knoten repräsentiert wird. Dabei steht ε für den leeren Kontext. Die Knoten sind in rot mit der dazugehörigen Wahrscheinlichkeitsverteilung (über dem Alphabet $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$) annotiert. Quelle: Schulz et al. (2008).

und Folgekontext eine Wahrscheinlichkeit zuordnet, so dass $\sum_{(\sigma,c')\in\Sigma\times C} \phi(c,\sigma,c') = 1$ für alle $c \in C$ gilt.

2.7 Definition. Ein allgemeines Textmodell ist durch (C, c_0, Σ, ϕ) gegeben, wobei C eine endliche Menge von Kontexten ist, $c_0 \in C$ der Startkontext, Σ ein endliches Alphabet und $\phi : C \times \Sigma \times C \to [0,1]$ für jedes $c \in C$ eine Wahrscheinlichkeitsverteilung auf $\Sigma \times C$ definiert.

Ähnliche Modelle werden von Kucherov et al. (2006) *probability transducers* genannt.

Das i.i.d.-Modell (M0) mit Parametervektor $p = (p_a)_{a\in\Sigma}$ bekommt man beispielsweise mit $C = \{\, c_0 \,\}$ (einelementige Zustandsmenge) und $\phi(c_0, a, c_0) := p_a$ für alle $a \in \Sigma$.

Durch die Entkopplung von Symbol und Kontext ist die Berechnung der Wahrscheinlichkeit, einen Text $s \in \Sigma^n$ zu erzeugen, nicht so einfach wie bisher: Es kann mehrere Kontextfolgen geben, die die gleiche Symbolfolge erzeugen. Der Symbolfolge können wir nicht ansehen, welche Kontextfolge sie erzeugt hat. Es sei C_i der Kontext vor der Emission von Symbol s_i, wobei $C_0 :\equiv c_0$ (Startkontext). Das Modell besagt, dass

$$\mathbb{P}(S = s_0, \ldots, s_{n-1},\, C = c_0, c_1, \ldots, c_n) = \prod_{i=0}^{n-1} \phi(c_i, s_i, c_{i+1}).$$

Wollen wir jetzt die Gesamtwahrscheinlichkeit für $s = s_0, \ldots, s_{n-1}$, müssen wir über alle Kontextfolgen summieren, das sind $|C|^n$ viele:

$$\mathbb{P}(S = s_0, \ldots, s_{n-1}) = \sum_{c_1,\ldots,c_n\in C^n} \prod_{i=0}^{n-1} \phi(c_i, s_i, c_{i+1}). \tag{2.1}$$

Glücklicherweise kann die Summe effizienter berechnet werden, als über alle $|C|^n$ Terme zu summieren: Sobald Kontext c_i bekannt ist, hängt die weitere Entwicklung von Symbolen und Kontexten nicht von der ferneren Vergangenheit, sondern nur von c_i ab (*Markov-Eigenschaft*). Daher definieren wir Hilfsgrößen $P(i, c)$ für $0 \leq i < n$ und $c \in C$:

$$P(i, c) := \mathbb{P}(S_0, \ldots, S_i = s_0, \ldots, s_i,\, C_{i+1} = c)$$

2.8 Lemma. Es ist $P(0, c) = \phi(c_0, s_0, c)$ für alle c. Für $i > 0$ ist $P(i, c) = \sum_{c'} P(i-1, c') \cdot \phi(c', s_i, c)$. Die gesuchte Wahrscheinlichkeit ist $\mathbb{P}_n(S = s) = \sum_c P(n-1, c)$.

Beweis. Die Aussage für $P(0, c)$ folgt direkt aus der Definition des Modells, der von ϕ und der von $P(0, c)$. Die Aussage für $P(i, c)$ folgt direkt aus (2.1), indem man die Summationsreihenfolge vertauscht und die Zwischenergebnisse in $P(i, c)$ abspeichert:

$$\sum_{c_1,\ldots,c_n \in C^n} \prod_{i=0}^{n-1} \phi(c_i, s_i, c_{i+1}) = \cdots \sum_{c_3} \underbrace{\sum_{c_2} \underbrace{\sum_{c_1} \underbrace{\phi(c_0, s_0, c_1)}_{P(0,c_1)}}_{P(1,c_2)} \phi(c_1, s_1, c_2)}_{P(2,c_3)} \phi(c_2, s_2, c_3) \cdots$$

Entsprechend ergibt sich für $\mathbb{P}_n(S = s)$ die letzte äußere Summe. Die Berechnung benötigt $\mathcal{O}(n|Q|^2)$ Rechenschritte. □

Dieser Algorithmus ist eng verwandt mit dem sogenannten Forward-Algorithmus bei Hidden Markov Modellen (Kapitel 3), der ebenfalls die Gesamtwahrscheinlichkeit für eine Symbolfolge berechnet, indem über mehrere Zustandsfolgen summiert wird. Der Formalismus unterscheidet sich nur geringfügig und man kann sogar zeigen, dass die hier definierten Textmodelle und die HMMs in Kapitel 3 äquivalent sind.

Bei den spezielleren Mk-Modellen war es stets so, dass man nicht über verschiedene Kontexte summieren musste, da jeweils die letzten Zeichen den Kontext eindeutig definierten.

2.7 Rechnen mit kleinen Wahrscheinlichkeiten*

Die beschränkte Genauigkeit von `double`-Fließkommazahlen (selbst bei erweiterter Genauigkeit) kann bei langen Berechnungen mit Wahrscheinlichkeiten Probleme verursachen. Zu beachten sind zwei Einschränkungen:

1. Es gibt eine kleinste darstellbare von Null verschiedene positive Zahl; diese liegt etwa bei 10^{-323}.
2. Die Zahl der Nachkommastellen ist beschränkt: Die Addition einer sehr kleinen Zahl ε zu einer anderen Zahl x muss nicht notwendigerweise zu einer Erhöhung von x führen. Für $x = 1$ ist $\varepsilon \approx 10^{-16}$ die kleinste Zahl, die man addieren kann, so dass $1 + \varepsilon > 1$. Dies ist dann problematisch, wenn durch eine Verkettung von Operationen eine eigentlich darstellbare Zahl herauskäme, die aber aufgrund der beschränkten Genauigkeit abgeschnitten wird. Beispiel mit $\varepsilon := 10^{-20}$: Bei numerischen Rechnungen ist $(1 + \varepsilon) - 1 = 1 - 1 = 0$; tatsächlich aber sollte natürlich ε herauskommen. Ebenso

ist numerisch $\log(1+\varepsilon) = \log(1) = 0$; tatsächlich sollte aber etwa ε herauskommen (Reihenentwicklung des Logarithmus). In beiden Beispielen zerstört die Addition einer großen Zahl (der 1) die vorhandene Information.

Da bei der Wahrscheinlichkeitsberechnung von Texten unter Textmodellen viele Wahrscheinlichkeiten multipliziert werden, tritt schnell der Fall ein, dass das Produkt unter die kleinste darstellbare positive Zahl fällt (und somit zu Null würde). Da die exakten Wahrscheinlichkeiten häufig unerheblich sind und es nur auf die Größenordnungen (Exponenten) ankommt, bieten sich zwei Lösungen an.

Zum einen kann man bei allen betrachteten Wahrscheinlichkeiten den größten Faktor ausklammern und nur dessen Exponenten speichern. Ein Vektor $p = [1.3 \cdot 10^{-500}, 2.3 \cdot 10^{-501}, 5.3 \cdot 10^{-507}]$ würde dann (bei Verwendung der Basis 10) als Paar $(-500, [1.3, 0.23, 0.00000053])$ gespeichert werden. Ein Nachteil ist, dass man für diese Darstellung eigene Datenstrukturen aufbauen muss.

Zum anderen kann man einfach direkt die Logarithmen der Wahrscheinlichkeiten speichern. Das hat den Vorteil, dass sich die Multiplikation zu einer Addition vereinfacht: Wollen wir $p = p_1 \cdot p_2$ im logarithmischen Raum berechnen, wird daraus einfach $\log p = \log p_1 + \log p_2$; die (zu kleinen) Werte p, p_1, p_2 treten dabei überhaupt nicht auf. Ein Nachteil ist, dass die Addition komplexer wird. Hierbei klammern wir den größten Summanden (dies sei p_1) aus, um daraus ein Produkt zu machen.

$$\begin{aligned} p &= p_1 + p_2 + \cdots + p_k \\ &= p_1(1 + p_2/p_1 + \cdots + p_k/p_1) \\ &= p_1(1+x) \\ \log p &= \log p_1 + \log(1+x) \\ &= \log p_1 + \mathrm{log1p}(x). \end{aligned}$$

Dabei ist log1p eine Funktion, die $\log(1+\cdot)$ berechnet, ohne explizit erst eine 1 zum Argument zu addieren, wenn dies zu einem Genauigkeitsverlust führen würde: Für $x \ll 1$ kann statt dessen eine geeignete Reihenentwicklung des Logarithmus verwendet werden. In Python ist diese Funktion als `math.log1p` implementiert. Wer nun wissen möchte, wie man diese Funktion (in C++) implementieren kann, sollte diesen Beitrag lesen: `http://www.johndcook.com/cpp_log_one_plus_x.html`.

Wir müssen noch klären, wie das Arugment x berechnet wird, da ja die p_i nur logarithmisch vorliegen. Es ist

$$\begin{aligned} x &= p_2/p_1 + \cdots + p_k/p_1 \\ &= \exp(\log p_2 - \log p_1) + \cdots + \exp(\log p_k - \log p_1) \end{aligned}$$

Diese Darstellung ist unproblematisch, da die Differenz $\log p_i - \log p_1$ so gering sein kann, dass die Exponentialfunktion davon problemlos darstellbar ist. Ist sie es nicht, dann ist p_i gegenüber p_1 so klein, dass sich die Summe dadurch ohnehin nicht verändern würde.

2.8 Parameterschätzung für Textmodelle

Wir kommen jetzt zu dem Problem, aus einem gegebenen Text Modellparameter zu einer gegebenen Modellklasse zu schätzen, also beispielsweise aus einer DNA-Sequenz die M0-Parameter $p = (p_A, p_C, p_G, p_T)$ zu schätzen.

Hierbei verwenden wir das *Maximum-Likelihood-Kriterium*, d.h., wir wollen die Parameter so wählen, dass unter allen Wahlmöglichkeiten die beobachtete Sequenz eine möglichst hohe Wahrscheinlichkeit bekommt.

Schätzung im M0-Modell Angenommen, es sind m Strings $s^j \in \Sigma^n$ gegeben, $j = 1 \dots m$. Gesucht sind die Parameter p_a, $a \in \Sigma$ unter der Bedingung $p_a \geq 0$ und $\sum_a p_a = 1$. Wir suchen die p_a, so dass die Gesamtwahrscheinlichkeit $\prod_{j=1}^{M} \mathbb{P}(s^j)$ maximal wird. Es ist $\prod_{j=1}^{m} \mathbb{P}(s^j) = \prod_{a\in\Sigma} p_a^{n_a}$ mit $n_a := \sum_{j=1}^{m} |\{i : s_i^j = a\}|$, also die Anzahl der Vorkommen von Zeichen a in allen m Strings. Logarithmieren führt auf das Optimierungsproblem

$$\begin{aligned} \text{Maximiere } & \sum_{a\in\Sigma} n_a \log p_a \\ \text{so dass } & \sum_{a\in\Sigma} p_a = 1. \end{aligned}$$

Es hat die eindeutige Lösung $p_a^* = n_a / \sum_b n_b$. Sie stimmt mit der intuitiven Vorstellung überein: p_a^* ist die relative Häufigkeit des Symbols a in den gegebenen Texten.

Aufgabe 2.1. Rechne nach, dass $p_a^* = n_a / \sum_b n_b$ tatsächlich die Maximum-Likelihood-Schätzung für p_a ist. (Optimierungsprobleme unter Nebenbedingungen kann man zum Beispiel mit Hilfe Lagrange'scher Multiplikatoren lösen.)

Schätzung im M1- und Mk-Modell. Analog schätzt man die Parameter im Markov-Modell (M1) oder allgemein im (Mk)-Modell: Bezeichnet n_y mit $y \in \Sigma^*$ die Gesamtzahl der Teilstrings y in allen Texten, dann ist die ML-Schätzung für $p_{x,a}$ gerade

$$p_{x,a}^* = n_{xa}/n_x \quad (x \in \Sigma^k,\ a \in \Sigma).$$

Bei geringen Beobachtungszahlen kann man das Modell durch *Pseudocounts* regularisieren.

2.9 Beispiel (Schätzung eines M1-Modells)**.** Auch wenn es nicht sinnvoll ist, ein Textmodell aus einem sehr kurzen Text zu schätzen, wollen wir dennoch tun um zu verdeutlichen, was dabei passiert. Sei $s =$ `abacab`. Zunächst ergibt sich als M0-Modell (durch Abzählen der Buchstabenhäufigkeiten) die Verteilung $p = (3/6, 2/6, 1/6)$. Durch Zählen der 2-mere erhalten wir die Matrix

$$P = \begin{pmatrix} 0 & 2/3 & 1/3 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

Dieses Beispiel verdeutlicht, dass für das robuste Schätzen eines Textmodells genügend Beobachtungen zur Verfügung stehen müssen. Hier ist das nicht der Fall: 5 Einträge in der Matrix sind 0, weil keine Beobachtungen vorliegen (nach `b` folgt immer `a`; nach `c` folgt immer `a`). ♡

Zur Schätzung der $p_{z,a}$ am Anfang der Sequenz ($|z| < k$) gibt es prinzipiell drei Möglichkeiten.

1. Sind viele verschiedene Sequenzen gegeben, ist die ML-Schätzung als relative Häufigkeit von a an der entsprechenden Position im entsprechenden Kontext möglich.
2. Eine Alternative ist, dass man die n_{xa} mit $|x| = k$ über die ersten Zeichen summiert:

$$p_{z,a} = \sum_{y \in \Sigma^{k-|z|}} n_{yza} / \sum_{y \in \Sigma^{k-|z|}} n_{yz}.$$

3. Eine andere Alternative, die numerisch zu ähnlichen, aber nicht identischen Ergebnissen wie die vorige Alternative führt, ist die $p_{z,a}$ so zu bestimmen, dass die Markovkette stationär wird.

Schätzung von Markov-Modellen variabler Ordnung. Wenn man die Ordnung für jeden Kontext (also die Trie-Topologie) fest vorgibt, ist die Schätzung mit relativen Häufigkeiten einfach möglich. Interessanter wird es, wenn man die (in einem zu definierenden Sinn) optimale Kontextlänge erkennen will. Dabei muss man sich nach zwei Kriterien richten:

1. Es müssen genug Beobachtungen für den entsprechenden Kontext vorhanden sein (sagen wir als Faustregel, mindestens $100|\Sigma|$), so dass die Verteilung des nächsten Zeichens sinnvoll bestimmt werden kann.
2. Für festes x sollten sich mindestens zwei Verteilungen der Kontexte ax ($a \in \Sigma$) hinreichend unterscheiden, sonst ist die Einbeziehung von a sinnlos.

Wir behandeln die Details an dieser Stelle nicht und verweisen auf die Arbeiten von Schulz et al. (2008).

Schätzung in allgemeinen Textmodellen mit endlichem Gedächtnis. Wenn der Kontext nicht eindeutig durch die Sequenzsymbole gegeben ist, wird die Schätzung optimaler Modellparameter schwierig. Die Likelihood-Funktion weist in der Regel viele lokale Maxima auf, so dass man auf iterative Verfahren zurückgreift, um zumindest ein solches lokales Maximum zu erreichen. Welches davon erreicht wird, hängt von den Startwerten für die Parameter ab, die man vorgeben muss. Wir kommen auf ein ähnliches Problem im Zusammenhang mit HMMs zurück und vertagen die Diskussion an dieser Stelle.

KAPITEL 3

Hidden-Markov-Modelle (HMMs)

3.1 Einführung

Bei der Analyse von Sequenzen (Audiosignalen in der Spracherkennung, Genomsequenzen in der Bioinformatik, Texten in der Linguistik) steht man häufig vor der Aufgabe, dass man aus einer Sequenz von Beobachtungen einen dahinterliegenden „Sinn“ erkennen möchte.

Wir geben ein einfaches Beispiel: Ein (nicht ganz ehrliches) Casino bietet ein Würfelspiel an: Der Einsatz beträgt einen Euro. Würfelt man eine Sechs, so erhält man 6 Euro zurück, in einem anderen Fall nichts. Auf den ersten Blick erscheint das Spiel fair. Das Casino verwendet jedoch zwei Würfel und wechselt diese nach jedem Wurf mit Wahrscheinlichkeit 1/10 aus. Der erste Würfel ist fair, der zweite würfelt 1–5 mit einer Wahrscheinlichkeit von jeweils 1/5 und niemals eine Sechs. Wir beobachten das Spiel 100 Würfe lang. Im Allgemeinen können wir nicht sagen, mit welchem Würfel ein bestimmter Wurf geworfen wurde (es ist schon erstaunlich genug, dass wir wissen, dass es einen gezinkten Würfel gibt, und nicht mal gegen das Casino vorgehen!), aber wir können langfristig sagen, welcher Würfel (bzw. welche Würfelfolge) die beobachtete Sequenz von Augenzahlen plausibel erzeugt haben könnte.

Formal ist ein HMM einfach eine Markovkette (mit endlich vielen Zuständen, im Beispiel den zwei Würfeln) mit bestimmten Übergangswahrscheinlichkeiten, und dazu gewissen Emissionen in jedem Zustand (im Beispiel sind das die Augenzahlen), die abhängig vom Zustand zufällig gezogen werden. Im Grunde haben wir HMMs schon als allgemeine Textmodelle mit endlichem Gedächtnis kennengelernt.

3.1 Definition (HMM)**.** Ein HMM besteht aus folgenden Komponenten:

- endliche Zustandsmenge Q
- Startzustand $q_0 \in Q$

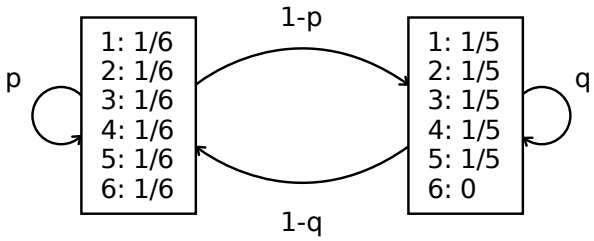


Abbildung 3.1: Hidden-Markov-Modell (HMM), das eine Folge von Würfen mit zwei verschiedenen Würfeln modelliert. Jeder Würfel wird durch einen Zustand repräsentiert. Dabei ist der eine Würfel fair, der andere unfair. Die Wahrscheinlichkeit, den Würfel zu wechseln beträgt $1 - p$ bzw. $1 - q$.

- Übergangswahrscheinlichkeiten $(a_{q',q})$; es ist $\sum_q a_{q',q} = 1$ für alle $q' \in Q$.
- Emissionsmenge E
- Emissionswahrscheinlichkeiten je Zustand $e_q(b)$ für $q \in Q$ und $b \in E$; es ist $\sum_b e_q(b) = 1$ für alle $q \in Q$.

Ein HMM startet in Zustand q_0, macht zunächst einen zufälligen Zustandsübergang gemäß der Übergangswahrscheinlichkeiten aus q_0, gelangt so in den ersten Zustand, wo gemäß Emissionswahrscheinlichkeiten ein Symbol ausgegeben wird.

Ein HMM liefert somit zwei stochastische Prozesse:

- Zustandsprozess/Pfad: $\Pi = \Pi_0\Pi_1\Pi_2 \ldots = (\Pi_t)$ (versteckt, engl. *hidden*)
- Emissionsprozess: $X = X_0X_1X_2 \ldots = (X_t)$ (beobachtet)

Dessen gemeinsames Wahrscheinlichkeitsmaß nennen wir wie immer $\mathbb{P}$.

Wir gehen davon aus, dass ein HMM n Schritte lang beobachtet wird. Aus dem Modell ergibt sich die Wahrscheinlichkeit für einen konkreten Pfad $\pi = (\pi_0, \ldots, \pi_{n-1}) \in Q^n$ zusammen mit einer Emissionssequenz (Beobachtung) $x = (x_0, \ldots, x_{n-1}) \in E^n$ als

$$\mathbb{P}(\Pi = \pi,\, X = x) = \prod_{t=0}^{n-1} a_{\pi_{t-1},\pi_t} \cdot e_{\pi_t}(x_t).$$

Dabei ist $\pi_{-1} := q_0$ immer der Startzustand; vor der ersten Emission findet bereits ein Übergang statt.

Zur Notation: $\mathbb{P}(\Pi = \pi,\, X = x)$ steht für die Wahrscheinlichkeit, dass der stochastische Pfadprozess Π genau dem (konkreten) Pfad π entspricht und dass gleichzeitig der Emissionsprozess X der (konkreten) Emissionsfolge x entspricht. Zur Abkürzung schreiben wir einfach $\mathbb{P}(\pi, x)$, obwohl das formal nicht ganz korrekt ist. Spätestens wenn wir nämlich auch $\mathbb{P}(X = x)$ einfach mit $\mathbb{P}(x)$ abkürzen ist nur noch am (beliebigen) Variablennamen erkennbar, ob wir einen Pfad oder eine Beobachtung meinen.

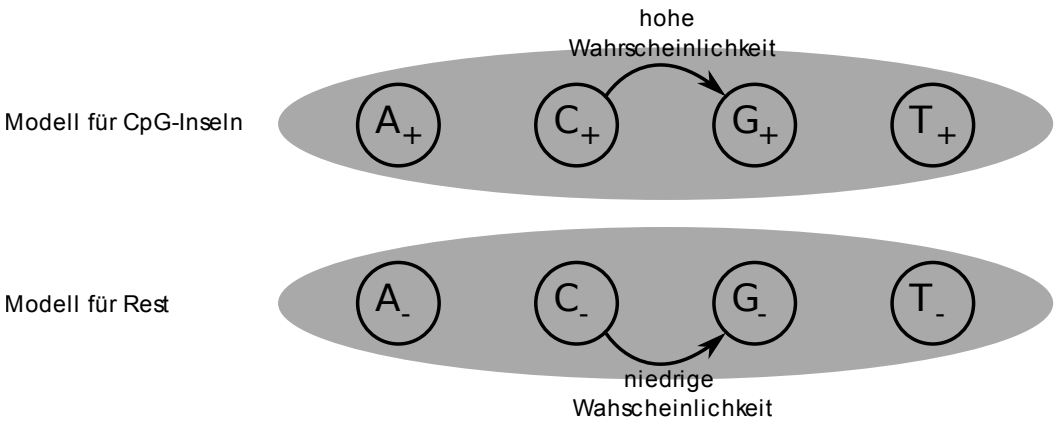


Abbildung 3.2: HMM zum Finden von CpG-Inseln in einem Genom. Nur zwei Kanten dieses vollständig verbundenen Graphen sind dargestellt, aber auch alle anderen Übergänge haben eine positive Warhscheinlichkeit. Diese Parameter sinnvoll zu schätzen, ist jedoch (besonders für die Kanten vom oberen zum unteren Teil und umgekehrt) nicht einfach.

3.2 Beispiele aus der Bioinformatik

Anwendung: CpG-Inseln. Im menschlichen Genom ist das Dinukleotid `CG` (also Cytosin gefolgt von Guanin, auch CpG geschrieben, das p deutet eine Phosphorgruppe an, die kalrstellen soll, dass es sich nicht um eine `C`≡`G`-Bindung auf komplementären Strängen der DNA handelt, sonderm um aufeinander folgende Nukleotide auf dem gleichen Strang!) unterrepräsentiert (d.h., es kommt selten vor) im Vergleich zu dem, was aufgrund der Häufigkeiten von `C` und `G` zu erwarten ist. CpG-Inseln sind Bereiche in einem Genom, in denen `CG` sehr häufig ist. Das Finden von CpG-Inseln kann mit Hilfe von HMMs erfolgen. Dazu definiert man zwei Textmodelle, eins für CpG-Inseln und eins für den Rest des Genoms. Diese beiden Modelle verbindet man zu einem HMM, das zwischen beiden Modellen wechseln kann. In Abbildung 3.2 ist ein solches Modell für zwei Textmodelle erster Ordnung dargestellt.

Finden von Open Reading Frames (ORFs). Ein *offener Leserahmen* (oder *open reading frame, ORF*) ist ein zusammenhängender Abschnitt in einem Genom, der möglicherweise ein Protein codiert, also mit einem Startcodon beginnt und mit einem Stoppcodon endet, nicht zu kurz und nicht zu lang ist (im Vergleich zu typischen Proteinlängen), und dessen Codon-Häufigkeiten nicht zu stark von den in diesem Organismus typsichen Codon-Häufigkeiten abweichen.

Man kann also ein HMM erstellen, dass aus mehreren „Bereichen“ besteht: einer für ORFs, wo zu Beginn ein Startcodon erzeugt wird, dann mit gewisser Wahrscheinlichkeit bestimmte Folgecodons, und zum Schluss ein Stoppcodon, sowie ein Bereich, der die sonstige genomische Sequenz modelliert. Die Verwendung von stummen Zuständen (siehe unten) erleichtert das Aufstellen des Modells.

3.3 Algorithmen auf HMMs

Drei Fragen sind für eine beobachtete Emissionssequenz $x \in E^n$ von Interesse.

1. Welche Wahrscheinlichkeit hat die gemachte Beobachtung? Berechne $\mathbb{P}(x) = \mathbb{P}(X = x) = \sum_{\text{Pfade } \pi} \mathbb{P}(\pi, x)$ (Formel von der totalen Wahrscheinlichkeit).
2. Welcher Pfad π maximiert $\mathbb{P}(\pi, x)$? Finde $\pi^* := \operatorname{argmax}_\pi \mathbb{P}(\pi, x)$
3. Berechne die bedingte Wahrscheinlichkeit $\mathbb{P}(\Pi_t = q \mid x)$. Finde den wahrscheinlichsten Zustand zu jedem Zeitpunkt, also $\hat{\pi}_t := \operatorname{argmax}_q \mathbb{P}(\Pi_t = q \mid x)$. Achtung: Häufig ist $\hat{\pi}_t \neq \pi_t^*$ (siehe unten).

Forward-Algorithmus: Berechnung der Wahrscheinlichkeit von x. Um Problem 1 zu lösen, könnten wir, wie es die Formel von der totalen Wahrscheinlichkeit suggeriert, die Summe über die Wahrscheinlichkeiten aller Pfade berechnen. Dann wäre die Laufzeit allerdings exponentiell. Statt dessen verfahren wir wie bei den Textmodellen und bedingen auf den letzten Zustand. Dazu definieren wir Hilfsvariablen, die sogenannten *Forward-Variablen*

$$f_q(t) := \mathbb{P}(X_{0\ldots t} = x_0, \ldots, x_t,\, \Pi_t = q) = \sum_{\pi_0 \ldots \pi_{t-1}} \mathbb{P}(x_0, \ldots, x_t,\, \pi_0 \ldots \pi_{t-1}, \Pi_t = q)$$

Dies ist die Wahrscheinlichkeit, bis zum Zeitpunkt (Schritt) t die gegebenen Beobachtungen zu machen und dabei im Zustand q zu landen. Summiert wird also über alle Pfad-Präfixe der Länge t, währen der Zustand zum Zeitpunkt t selbst festgesetzt wird. Der *Forward-Algorithmus* besteht darin, diese Größen für zunehmende t auszurechnen.

3.2 Lemma (Forward-Algorithmus)**.** Es ist

$$f_q(0) = \mathbb{P}(x_0, \Pi_0 = q) = a_{q_0,q} e_q(x_0) \qquad \text{für alle } q \in Q,$$

$$f_q(t) = \left(\sum_{q'} f_{q'}(t-1) \cdot a_{q',q} \right) \cdot e_q(x_t) \qquad \text{für } 1 \leq t \leq n-1,\, q \in Q.$$

Zum Schluss berechnen wir $\mathbb{P}(x) = \sum_{q \in Q} f_q(n-1)$.

Der Beweis erfolgt wie bei den Textmodellen: Es werden lediglich Teilsummen in den $f_q(t)$ „zwischengespeichert“; per Induktion wird gezeigt, dass tatsächlich korrekt die Summe über alle Pfade berechnet wird.

Viterbi-Algorithmus: Berechnung des Maximum-Likelihood-Pfads. Jetzt lösen wir das zweite Problem und suchen nach dem Pfad π^*, der die gemeinsame Wahrscheinlichkeit $\mathbb{P}(\pi, x)$ über alle π maximiert, sowie nach dem Wert dieser Wahrscheinlichkeit.

Auch hier bietet sich ein schrittweises Vorgehen an, analog zum Forward-Algorithmus. Wir definieren die sogenannten *Viterbi-Variablen*

$$v_q(t) := \max_{\pi_0 \ldots \pi_{t-1}} \mathbb{P}(X_{0\ldots t} = x_0, \ldots, x_t,\, \Pi_{0\ldots t-1} = \pi_0 \ldots \pi_{t-1}, \Pi_t = q).$$

Hier ist über alle Pfad-Präfixe der Länge t maximiert worden. Der *Viterbi-Algorithmus* besteht darin, diese Größen für zunehmende t auszurechnen.

3.3 Lemma (Viterbi-Algorithmus)**.**

$$\begin{aligned} v_q(0) &= \mathbb{P}(x_0, \Pi_0 = q) = a_{q_0,q} e_q(x_0) && \text{für alle } q \in Q, \\ v_q(t) &= \max_{q'} \left(v_{q'}(t-1) \cdot a_{q',q}\right) \cdot e_q(x_t) && \text{für } 1 \leq t \leq n-1,\ q \in Q. \end{aligned}$$

Zum Schluss berechnen wir $\mathbb{P}(x, \pi^*) = \max_{q \in Q} v_q(n-1)$.

Beweis. Trivialerweise gilt

$$v_q(t) \geq \max_{q'} \left(v_{q'}(t-1) \cdot a_{q',q}\right) \cdot e_q(x_t),$$

da auf der rechten Seite über eine Auswahl an Pfaden maximiert wird, die in Schritt t in q enden. Interessant ist die andere Richtung der Ungleichung. Wir führen den Beweis per Induktion und Widerspruch. Der Induktionsanfang $v_q(0)$ (s.o.) ist klar; sei also $t \geq 1$. Angenommen, es gäbe einen Pfad $\pi^* = \pi_0^*, \ldots \pi_{t-1}^*$, so dass

$$\begin{aligned} v_q(t) &\equiv \max_{\pi_0 \ldots \pi_{t-1}} \mathbb{P}(x_0, \ldots, x_t,\ \pi_0 \ldots \pi_{t-1}, \Pi_t = q) \\ &= \mathbb{P}(x_0, \ldots, x_t,\ \pi_0^* \ldots \pi_{t-1}^*, q) \\ &> \max_{q'} \left(v_{q'}(t-1) \cdot a_{q',q}\right) \cdot e_q(x_t). \end{aligned}$$

Ein solcher Pfad muss zum Zeitpunkt $t-1$ in irgendeinem Zustand q^* sein. Beim Verlängern nach Zustand q zum Zeitpunkt t kommt genau der Faktor $a_{q^*,q} e_q(x_t)$ zur bereits berechneten Wahrscheinlichkeit hinzu; genau dieser Faktor steht aber auch auf der rechten Seite für $q' = q^*$. Es wäre also nur möglich, dass bereits der Pfad π^* bis $t-1$ schon eine höhere Wahrscheinlichkeit gehabt hätte als $v_{q^*}(t-1)$; dies ist jedoch ein Widerspruch zur Induktionsvoraussetzung. □

Auf die genannte Weise erhalten wir die Wahrscheinlichkeit von x entlang des besten Pfades, aber wie erhalten wir den Pfad? Den bekommen wir, indem wir uns in jedem Schritt merken, welcher Zustand das Maximum erreichte: Zu jedem q und t tabellieren wir $\phi_q(t) := \operatorname{argmax}_{q'} (v_{q'}(t-1) \cdot a_{q',q})$.

Dann ist das Ende des Pfades $\pi_{n-1}^* := \operatorname{argmax}_q v_q(n-1)$, und $\pi_{t-1}^* := \phi_{\pi_t^*}(t)$ für $1 \leq t \leq n-1$.

Dies nennt man ein *Traceback*[1]. Im Grunde ist die explizite Speicherung der $\phi_q(t)$ nicht nötig, wenn wir bereits die $v_q(t)$ speichern. Wir könnten ja einfach erneut ausrechnen, welcher Zustand q' jeweils zu einer Maximierung der Wahrscheinlichkeit führt. Aber da wir dies ohnehin schon berechnet haben, gibt es keinen Grund, die gleiche Arbeit mehrfach zu machen. Der so berechnete Pfad wird *Viterbi-Pfad* oder *optimaler Pfad* genannt.

Der Zeitaufwand ist offensichtlich $\mathcal{O}(|Q|^2 n)$, der Speicherbedarf beträgt nur $\mathcal{O}(|Q|)$ für die $v_q(t)$, da wir gleichzeitig immer nur v_q für das aktuelle t und für $t-1$ benötigen. Für das Traceback benötigen wir jedoch $\mathcal{O}(|Q|n)$ Platz. Mit einem Divide-and-Conquer-Trick lässt sich dieser (durch eine Erhöhung der Laufzeit) allerdings auf $\mathcal{O}(|Q|)$ drücken, worauf wir hier nicht eingehen.

Wir sehen, dass Forward- und Viterbi-Algorithmus im Grunde identisch sind. Die wesentlichen Unterschiede sind: Bei Viterbi betrachten wir Maxima statt Summen, und uns interessiert meist eher der Traceback als die maximale Wahrscheinlichkeit selbst.

[1] Bitte Traceback und Backtracing niemals mit Backtracking verwechseln; das ist etwas völlig anderes.

Wie gut ist der optimale Pfad? Vergleiche $\mathbb{P}(x, \pi^*)$ und $\mathbb{P}(x)$:

$$\mathbb{P}(x) = \sum_{\pi} \mathbb{P}(x, \pi) = \mathbb{P}(x, \pi^*) + \sum_{\pi \neq \pi^*} \mathbb{P}(x, \pi) \overset{\text{(i)}}{\geq} \mathbb{P}(x, \pi^*)$$

Dabei gilt für (i) ungefähr Gleichheit, wenn $\mathbb{P}(x, \pi)/\mathbb{P}(x, \pi^*) \approx 0$ für alle $\pi \neq \pi^*$, d.h. dann, wenn es nur einen sinnvollen Pfad gibt. Dies kommt aber in der Praxis selten vor. In der Regel gibt es mehrere etwa gleich gute Pfade, die durchaus unterschiedlicher „Meinung“ sein können, in welchem Zustand die Beobachtung $X_t = x_t$ gemacht wurde.

Forward-Backward-Algorithmus zur Berechnung von $\mathbb{P}(\Pi_t = q \mid x)$. Da wir die gemeinsame Verteilung von Zustandspfad und Beobachtung kennen, und x beobachtet haben, können wir die bedingte Verteilung von Π_t gegeben x ausrechnen: Es ist

$$\begin{aligned}\mathbb{P}(\Pi_t = q \mid x) &= \mathbb{P}(\Pi_t = q, x) \,/\, \mathbb{P}(x) \\ &= \mathbb{P}(\Pi_t = q,\, x_0, \ldots, x_t) \cdot \mathbb{P}(x_{t+1}, \ldots, x_{n-1} \mid \Pi_t = q, x_0, \ldots, x_t) \,/\, \mathbb{P}(x) \\ &= \mathbb{P}(\Pi_t = q, x_0, \ldots, x_t) \cdot \mathbb{P}(x_{t+1}, \ldots, x_{n-1} \mid \Pi_t = q) \,/\, \mathbb{P}(x),\end{aligned}$$

da die Emissionen ab $t+1$ bei gegebenem Π_t unabhängig von den vergangenen Emissionen sind. Der erste Faktor ist uns bereits als Forward-Variable $f_q(t)$ bekannt. Sinnigerweise nennen wir den zweiten Faktor nun *Backward-Variable*

$$b_q(t) := \mathbb{P}(x_{t+1}, \ldots, x_{n-1} \mid \Pi_t = q).$$

Genau wie die Forward-Variablen lassen sich die Backward-Variablen sukzessive ausrechnen (Lemma 3.4, siehe unten); im Unterschied zu den Forward-Variablen geschieht dies allerdings für fallende t; daher auch die Namensgebung.

Wir haben dann

$$\begin{aligned}\mathbb{P}(\Pi_t = q \mid x) &= f_q(t) \cdot b_q(t) \,/\, \mathbb{P}(x) \\ &= f_q(t) \cdot b_q(t) \,/\, \sum_{q'} f_{q'}(t) b_{q'}(t); \qquad (3.1)\end{aligned}$$

die Gesamtwahrscheinlichkeit $\mathbb{P}(x)$ kann dabei entweder aus dem Forward-Algorithmus oder durch die hier angegebene Summe über alle Zustände q' berechnet werden. Wir kommen nun zur Berechnung der $b_q(t)$.

3.4 Lemma (Backward-Algorithmus)**.**

$$\begin{aligned} b_q(n-1) &= \mathbb{P}(\{\ \} \mid \Pi_{n-1} = q) = 1 && \text{für } q \in Q; \\ b_q(t) &= \sum_{q'} a_{q,q'} \cdot e_{q'}(x_{t+1}) \cdot b_{q'}(t+1) && \text{für } n-1 > t \geq 0,\ q \in Q. \end{aligned}$$

Beweis. Der Fall $t = n - 1$ ist klar. Für $0 \leq t < n - 1$ ist

$$\begin{aligned} b_q(t) &= \mathbb{P}(x_{t+1}, \ldots, x_{n-1} \mid \Pi_t = q) \\ &= \sum_{q'} \mathbb{P}(x_{t+1}, \ldots, x_{n-1}, \Pi_{t+1} = q' \mid \Pi_t = q) \\ &= \sum_{q'} \mathbb{P}(\Pi_{t+1} = q' \mid \Pi_t = q) \cdot \mathbb{P}(x_{t+1}, \ldots, x_{n-1} \mid \underline{\Pi_t = q}, \Pi_{t+1} = q') \\ &= \sum_{q'} a_{q,q'} \cdot \mathbb{P}(x_{t+1} \mid \Pi_{t+1} = q') \cdot \mathbb{P}(x_{t+2}, \ldots, x_{n-1} \mid \underline{x_{t+1}}, \Pi_{t+1} = q') \\ &= \sum_{q'} a_{q,q'} \cdot e_{q'}(x_{t+1}) \cdot b_{q'}(t+1), \end{aligned}$$

wobei die unterstrichenen Terme in den Bedingungen aufgrund der Unabhängigkeitsannahmen im Modell gestrichen werden konnten. □

Die hier vorgestellte Berechnung der *a-posteriori-Verteilung* der Zustände $\mathbb{P}(\Pi_t = q \mid x) = f_q(t) \cdot b_q(t)/\mathbb{P}(x)$ bezeichnet man auch als *posterior decoding*. Die Berechnung mit Hilfe der Forward- und Backward-Wahrscheinlichkeiten bezeichnet man auch als *Forward-Backward-Algorithmus*.

Wir können nun den wahrscheinlichsten Zustand $\hat{\pi}_t$ zum Zeitpunkt t finden:

$$\hat{\pi}_t := \operatorname*{argmax}_q f_q(t) \cdot b_q(t).$$

Die Folge der $\hat{\pi}_t$ gibt für jeden Zeitpunkt den wahrscheinlichsten Zustand an, ist aber nicht notwendiger Weise selbst ein gültiger Pfad.

Man kann die Folge der $\hat{\pi}_t$ mit dem Viterbi-Pfad π^*_t vergleichen. Stimmen diese für ein bestimmtes t überein, gehen alle wichtigen Pfade durch diesen Zustand und der Viterbi-Pfad ist eine Vorhersage, in die man eine relativ hohe Konfidenz haben kann. In der Tat kann man die Konfidenz direkt durch die a-posteriori-Wahrscheinlichkeit $\mathbb{P}(\Pi_t = \pi^*_t \mid x)$ angeben. Stimmen $\hat{\pi}_t$ und π^*_t nicht überein, ist der dieser Viterbi-Vorhersage keine hohe Konfidenz zu geben.

3.4 Training (Parameterschätzung)

Da HMMs nicht "vom Himmel fallen", müssen ihre Parameter in der Regel aus gegebenen Beobachtungen trainiert werden. Je nachdem, ob dazu nur die Emissionen oder auch die Zustandsabfolgen (Pfade) geben sind, ist das Training einfach bis sehr schwierig.

3.4.1 Training bei bekannten Pfaden

Wenn die Pfade bekannt sind, besteht das Training der Übergangs- und Emissionswahrscheinlichkeiten im wesentlichen aus Zählen. Die Emissionsverteilung $e_q = (e_q(b))_{b \in \Sigma}$ eines Zustands q wird geschätzt aus der relativen Häufigkeit der einzelnen Emissionen b, wenn der Pfad den Zustand q benutzt. Die Übergangswahrscheinlichkeiten aus Zustand q werden

analog durch Zählen der Übergänge in die Folgezustände bestimmt. Um Nullwahrscheinlichkeiten zu vermeiden, kann man wieder mit Pseudocounts arbeiten, die man zu den Beobachtungen hinzuzählt. Bei großen Datenmengen üben diese kaum noch einen Einfluss aus, bei kleinen Datenmengen simulieren sie die Gleichverteilung (Unwissenheit).

3.4.2 Training ohne bekannte Pfade (Baum-Welch)

Das eigentliche Ziel des Training besteht darin, die HMM-Parameter $\theta = ((e_q(b)), (a_{q',q}))$ (Emissions- und Übergangswahrscheinlichkeiten in jedem Zustand) zu finden, die bei N beobachteten Sequenzezn $x^{(1)}, \ldots, x^{(N)}$ die Wahrscheinlichkeit, eben jene Beobachtung zu machen, maximieren: Finde

$$\theta^* := \operatorname*{argmax}_\theta \sum_{i=1}^{N} \log \mathbb{P}_\theta(X = x^{(i)})$$

Tatsächlich ist dieses Problem nicht einfach lösbar, und es gibt viele lokale Optima. (Sofern man keinen festen Startzustand annimmt, muss auch noch die Startverteilung der Zustände geschätzt werden.)

Folgender iterativer Prozess, der sogenannte "*Baum-Welch-Algorithmus*" funktioniert (bei geegneten Startwerten) in der Praxis gut und konvergiert gegen ein lokales Optimum; er ist eine Variante des EM-Algorithmus (expectation maximization). Vorausgesetzt wird, dass bereits initiale Modellparameter θ^0 existieren; diese werden iterativ verbessert. Diese initialen Parameter erhält man meist durch sinnvolles Einbringen von Vorwissen; eine rein zufällige Initialisierung führt in der Regel zu nicht interpretierbaren Modellen und schlechten lokalen Optima.

Die Idee ist wie folgt: Sind Modellparameter bekannt, lassen sich mit dem Forward-Backward-Algorithmus a-posteriori Wahrscheinlichkeiten für Aufenthaltswahrscheinlichkeiten in jedem Zustand zu jedem Zeitpunkt bestimmen. Gleichermaßen lassen sich auch a-posteriori-Übergangswahrscheinlichkeiten zwischen Zuständen zu jedem Paar von aufeinanderfolgenden Zeitpunkten bestimmen. Diese aggregiert man über alle Zeitpunkte und erhält so neue Schätzungen für die Übergangswahrscheinlichkeiten. Für die Emissionswahrscheinlichkeiten kombiniert man Zustands-Aufenthaltswahrscheinlichkeiten mit den beobachteten Emissionen.

Im Detail: Wir hatten bereits mit den Forward- und Backward-Variablen die bedingte Wahrscheinlichkeit

$$\mathbb{P}(\Pi_t = q \mid x) = f_q(t) \cdot b_q(t) \,/\, \mathbb{P}(x) = f_q(t) \cdot b_q(t) \,/\, \sum_{q'} f_{q'}(t) b_{q'}(t)$$

ausgerechnet (bei gegebener Beobachtung x die Aufenthaltswahrscheinlichkeit in q zum Zeitpunkt t). Daraus lassen sich neue Emissionswahrscheinlichkeiten $e_q^+(v)$ wie folgt schätzen (das Superskript $^+$ soll ein Update bzw. die nächste Iteration andeuten): Wir berechnen den Anteil der Gesamtaufenthaltsmasse in q, bei der ein v emittiert wird, durch Aggregation über die Zeitpunkte:

$$e_q^+(v) = \frac{\sum_{t:\, x_t = v} \mathbb{P}(\Pi_t = q \mid x)}{\sum_t \mathbb{P}(\Pi_t = q \mid x)}.$$

Zu beachten ist, dass im Zähler nur über die Zeitpunkte summiert wird, bei denen ein v beobachtet wird, während im Nenner über alle Zeitpunkte summiert wird.

Zur Schätzung der Übergangswahrscheinlichkeiten benötigen wir die *gemeinsame Zustandsverteilung* (gegeben die Beobachtung x) $\mathbb{P}(\Pi_t = q', \Pi_{t+1} = q \mid x)$ zu zwei aufeinander folgenden Zeitpunkten t und $t+1$. Dann können wir

$$a^+_{q',q} = \frac{\sum_{t=0}^{n-2} \mathbb{P}(\Pi_t = q',\, \Pi_{t+1} = q \mid x)}{\sum_{t=0}^{n-2} \mathbb{P}(\Pi_t = q' \mid x)}$$

schätzen. Die Wahrscheinlichkeiten im Nenner sind uns bekannte (a-posteriori-Zustandsverteilungen), aber auch die Wahrscheinlichkeiten im Zähler lassen sich mit Hilfe der Forward- und Backward-Variablen ausdrücken:

$$\mathbb{P}(\Pi_t = q',\, \Pi_{t+1} = q \mid x) = \frac{f_{q'}(t) \cdot a_{q',q} \cdot e_q(x_{t+1}) \cdot b_q(t+1)}{\sum_{p' \in Q, p \in Q} f_{p'}(t) \cdot a_{p',p} \cdot e_p(x_{t+1}) \cdot b_p(t+1)}.$$

(Hierbei wird im Vergleich zur Forward-Backward-Methode (3.1) ein Übergangsschritt und eine Emission in der Terme im Zähler und Nenner intergriert. Im Nenner wird über alle Zustandspaare summiert; dies ist für alle (q', q) dieselbe Normalisierungskonstante.)

Während diese Methode erst einmal relativ intuitiv anmutet, ist durch diese Beschreibung nicht klar, dass durch einen Iterationsschritt irgend etwas verbessert wird. Man kann tatsächlich zeigen (mit Hilfe allgemeiner Aussagen zum sogenannten EM-Algoritmus), dass in jedem Iterationsschritt die Gesamt-Likelihood des Modells erhöht wird und die Parameter gegen ein lokales(!) Optimum konvergieren. Wir gehen an dieser Stelle auf den Beweis nicht weiter ein.

3.5 Erweiterungen

Bisher haben wir HMMs und ihre Algorithmen in der “Basis”-Variante kennengelernt. In diesem Abschnitt erweitern wir HMMs um verschiedene für die Praxis sehr wichtige Möglichkeiten zur Modellierung.

3.5.1 Feste und modellbestimmte Textlängen

Bisher sind wir davon ausgegangen, dass ein HMM für eine feste Dauer von n Schritten „läuft“. Formal heißt das, dass die stochastischen Prozesse der Emissionen (X_t) und der Zustände (Π_t) potenziell unendlich lang sind, aber immer nur ein Präfix der Länge n betrachtet wird.

Wie bei Textmodellen ist es aber auch möglich und sinnvoll, das Modell selbst entscheiden zu lassen, wann der Prozess endet. Dies kann formal geschehen, indem ein absorbierender neuer Zustand (nennen wir ihn `$`) eingeführt wird, der mit Wahrscheinlichkeit 1 ein Text-Ende-Zeichen, das nicht im sonstigen Emissionsalphabet enthalten ist, ausgibt (etwa wiederum mit `$` bezeichnet). Sobald dieser Zustand erreicht wurde, verbleiben wir mit Wahrscheinlichkeit 1 darin. In den „normalen“ Zuständen ist die Emissionswahrscheinlichkeit für `$` null.

Die Wahrscheinlichkeit, genau die Sequenz $x_0, \ldots, x_{n-1}$ auszugeben (und nicht mehr!), ist dann gleich der Wahrscheinlichkeit, in $n+1$ Schritten die Sequenz $x_0, \ldots, x_{n-1}$,\$ auszugeben, und die bestehenden Algorithmen können weiter verwendet werden.

Die Wahrscheinlichkeit, dass die Länge des ausgegebenen Textes $\leq n$ ist, ist durch die Wahrscheinlichkeit $\mathbb{P}(X_n = \$) = \mathbb{P}(\Pi_n = \$)$ gegeben. Dies kann mit zum Forward-Algorithmus ähnlichen Algorithmen berechnet werden.

3.5.2 Stumme Zustände

Um HMMs modular aufzubauen und trotzdem die Anzahl der Parameter gering zu halten, ist es zweckmäßig, *stumme Zustände* einzuführen, also Zustände, die kein Symbol emittieren. Diese können als Beginn und Abschluß modularer Bereiche verwendet werden, so dass der Ein- und Austritt in ein Teil-HMM nur über diese speziellen Zustände möglich ist. Dadurch spart man sich Kanten zwischen Zuständen aus verschiedenen Modulen und reduziert damit die Parameterzahl des Modells.

Als Beispiel kann das CpG-Insel-Modell aus Abbildung 3.2 dienen: Da Übergänge zwischen den Teilmodellen „+“ und „-“'ohnehin selten sind, macht es wenig Sinn, eine eigene Kante zwischen jedem Zustandspaar zu haben. Stattdessen fügt man einen stummen Zustand ein, aus dem das „+“-Modell verlassen und das „-“-Modell betreten wird, und einen weiteren für die Gegenrichtung.

Stumme Zustände sind verwandt mit ε-Transitionen bei NFAs: Sie erlauben einen Zustandswechsel ohne Lesen (oder Emittieren) eines Symbols.

Das Problem bei stummen Zuständen ist, dass die HMM-Algorithmen bereits bei der Definition der Hilfsvariablen (Viterbi-, Forward-, Backward-Variablen) voraussetzen, dass bei jedem Schritt auch ein Symbol emittiert wird. Die Algorithmen müssen also für stumme Zustände angepasst werden.

Eine Voraussetzung dabei ist, dass es im HMM keine Zyklen gibt, die nur aus stummen Zuständen bestehen (in diesen könnte man endlos rotieren). Somit lassen sich also die stummen Zustände so anordnen, dass für jede Kante $q' \to q$ zwichen zwei stummen Zuständen q', q in der Anordnung immer $q' < q$ gilt. Man spricht auch von einer topologischen Sortierung der stummen Zustände.

Forward-Algorithmus. Beim Forward-Algorithmus ändert sich die Interpretation der Forward-Variablen: Wir definieren nun $f_q(t)$ als die Wahrscheinlichkeit, mit t Emissionen die Symbolfolge $x_1, \ldots, x_t$ zu erzeugen, so dass bei oder nach der t-ten Ausgabe (aber vor der nächsten Ausgabe) der Zustand q besucht wird. Dies lässt sich mit unserer Notation im Moment nicht formal hinschreiben, da wir Pfad und Emissionen mit parallel laufendem Index definiert haben.

Im Gegensatz zu vorher gilt nicht mehr, dass $\mathbb{P}(x_1, \ldots, x_t)$ die Summe $\sum_q f_q(t)$ ist. Dies gilt nur dann, wenn man die Summe auf emittierende Zustände einschränkt.

Für emittierende (nicht stumme) Zustände q werden die Forward-Variablen $f_q(t)$ normal berechnet. Darauf folgend werden die stummen Zustände in topologischer Sortierung abgearbeitet. Dort setzen wir $f_q(t) = \sum_{q'} f_{q'}(t)\, a_{q',q}$. Dabei läuft q' über alle Vorgängerzustände

von q (ob stumm oder nicht), und es ist sichergestellt, dass stumme q' bereits vorher bearbeitet worden.

Viterbi-Algorithmus. Beim Viterbi-Algorithmus ändert sich die Interpretation analog: Es sei $v_q(t)$ die maximale Wahrscheinlichkeit entlang eines Pfades, mit t Emissionen die Symbolfolge $x_1, \ldots, x_t$ zu erzeugen und bei oder nach der t-ten Emission (aber vor der nächsten) den Zustand q besucht zu haben. Für nicht stumme Zustände werden die Viterbi-Variablen $v_q(t)$ wie gehabt berechnet. In stummen Zuständen (in topologischer Sortierung) setzen wir $v_q(t) = \max_{q'} v_{q'}(t)\, a_{q',q}$.

Bei der Definition des Backtraces $\phi_q(t)$ ist darauf zu achten, dass emittierende Zustände stets auf emittierende Zustände verweisen. Dies lässt sich einrichten, indem stumme Zustände auf den emittierenden Vorgängerzustand verweisen und eventuell auf stumme Zustände verweisende $\phi_q(t)$ direkt weitergeleitet werden.

Backward-Algorithmus. Die Interpretation ist hier weniger intuitiv, aber analog zur Änderung beim Forward-Algorithmus: hier ist $b_q(t)$ die Wahrscheinlichkeit, $x_{t+1}, \ldots, x_{n-1}$ zu erzeugen unter der Bedingung, dass bei oder nach der t-ten Ausgabe (aber vor der $(t+1)$-ten) der Zustand q besucht wurde.

Die Werte der emittierenden Zustände q werden mit der normalen Formel berechnet. Nun müssen die stummen Zustände in umgekehrter topologischer Sortierung bearbeitet werden. Dort gilt $b_q(t) = \sum_{q'} a_{q,q'} \cdot b_{q'}(t+1)$.

3.5.3 Verweildauer in Zuständen

Wir betrachten das Beispiel eines HMMs mit zwei Zuständen. Dabei modelliert der eine Zustand (F) einen fairen, der andere Zustand (U) einen unfairen Würfel (siehe Abbildung 3.1). Wir stellen die Frage, wie lange das Modell in einem der Zustände verweilt. Im Folgenden bezeichnen wir die Verweildauer im Zustand F mit T und das Wahrscheinlichkeitsmaß, das sich auf eine Zustandsfolge bezieht, die im Zustand F beginnt, mit $\mathbb{P}$. Dann gilt:

$$\mathbb{P}(T \geq t) = p^{t-1} \qquad (t \geq 1)$$
$$\mathbb{P}(T = t) = p^{t-1} \cdot (1-p) \qquad (t \geq 1)$$

Für die Verweildauer in einem Zustand ergibt sich also eine geometrische Verteilung. Es gibt jedoch Anwendungen, in denen dies unvorteilhaft ist. Wie ist es möglich, ein Modell mit beliebigen Verteilungen von Verweildauern zu realisieren?

Ansätze:

- Speichere zu jedem Zustand die bisherige Verweildauer bis zu einer maximal möglichen Verweildauer $T_{\max}$. Nachteil: Dies erhöht die Komplexität aller Algorithmen um den Faktor $T_{\max}$. Äquivalent dazu: Jeder Zustand wird T_{max}-mal dupliziert (siehe Abbildung 3.3).

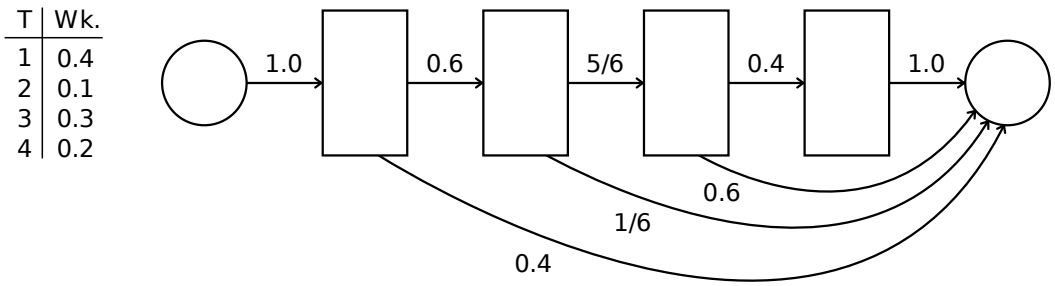


Abbildung 3.3: Beispiel für einen Teil eines HMMs, das mit Hilfe mehrerer Zustände eine beliebige vorgegebene Verteilung der Verweildauer T realisiert. Zur Begrenzung des Moduls werden stumme Zustände verwendet.

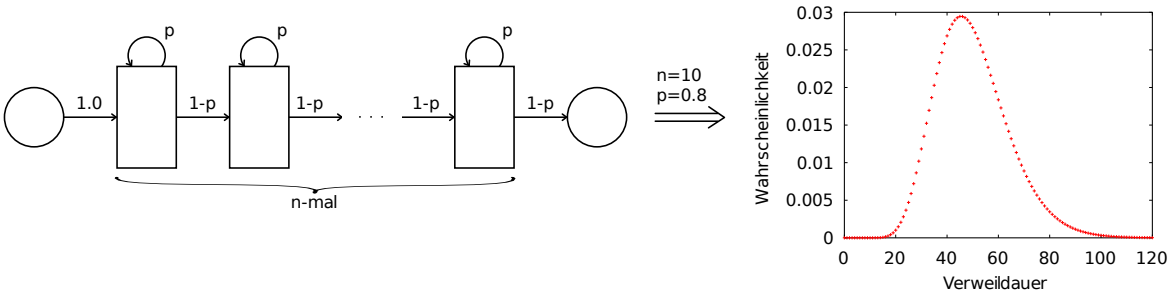


Abbildung 3.4: Teil eines HMMs mit einer unimodalen Verteilung der Verweildauer. Je größer n, desto mehr nähert sich die Verteilung einer Normalverteilung (zentraler Grenzwertsatz). Die Emissionen folgen in jedem der inneren Zustände der gleichen Verteilung und sind hier nicht relevant. Zur Begrenzung des Moduls werden stumme Zustände verwendet.

- Durch Hintereinanderschalten von n Zuständen mit Selbstübergängen (deren Verweildauer daher jeweils geometrisch verteilt ist) lässt sich für große n eine Verteilung realisieren, die einer Normalverteilung nahe kommt. Allerdings ist die Verteilung links abgeschnitten, d.h. die minimale Verweildauer beträgt mindestens n. Eine solche Verteilung nennt man negative Binomialverteilung oder diskrete Gammaverteilung. Bei n Zuständen mit jeweils Schleifenwahrscheinlichkeit p ergibt sich damit die Wahrscheinlichkeit, insgesamt genau k Schritte zu benötigen, als $\binom{n+k-1}{n-1} p^{k-n}(1-p)^n$; siehe Abbilung 3.4. Der Erwartungswert beträgt $n/(1-p)$.
- Durch Parallelschalten von Kopien der vorherigen Konstruktion mit jeweils unterschiedlichen Parametern und Gewichten lässt sich eine Mischung von unimodalen Verteilungen (und daher eine multimodale Verteilung) realisieren.

KAPITEL 4

Positions-Gewichts-Matrizen

In diesem Abschnitt lernen wir eine Möglichkeit kennen, eine (potenziell sehr große) Menge von Strings kompakt mit einem Muster in Form einer Positions-Gewichts-Matrix zu beschreiben (PWM, auch: positions-spezifische Scorematrix, PSSM).

Als Motivation dienen uns Transkriptionsfaktorbindestellen der DNA. Transkriptionsfaktoren (Proteine, die die Transkription von DNA beeinflussen; TFs) binden physikalisch an bestimmte Stellen der DNA. Diese Stellen heißen konsequenterweise Transkriptionsfaktorbindestellen (TFBSs). Sie lassen sich durch das dort vorhandene Sequenzmotiv charakterisieren. Zum Beispiel binden die TFs der „Nuclear factor I“ (NF-I) Familie als Dimere an das DNA-IUPAC-Motiv `5'-TTGGCNNNNNGCCAA-3'`; dabei steht `N` für ein beliebiges Nukleotid. Viele solcher Bindemotive, insbesondere wenn das Protein als Dimer bindet, sind identisch zu ihrem reversen Komplement. Ein Bindemotiv ist aber nicht exakt zu interpretieren. Schon an diesem einen Beispiel sieht man, dass bestimmte Positionen nicht exakt vorgegeben sein müssen. Manche Stellen sind variabler als andere, bzw. die Stärke der Bindung verringert sich um so stärker, je unterschiedlicher die DNA-Sequenz zum Idealmotiv ist.

4.1 Definition vom PWMs

Wie beschreibt man das Idealmotiv einer TFBS und die Unterschiede dazu? Man könnte einfach die Anzahl der unterschiedlichen Symbole zählen. Es stellt sich aber heraus, dass dieses Vorgehen zu grob ist. Statt dessen gibt man jedem Nukleotid an jeder Position eine Punktzahl (Score) und setzt beispielsweise einen Schwellenwert fest, ab dem eine hinreichend starke Bindung nachgewiesen wird. Alternativ kann man mit Methoden der statistischen Physik den Score-Wert in eine Bindungswahrscheinlichkeit umrechnen. Als Modell für Transkriptionsfaktorbindestellen haben sich daher Positions-Gewichts-Matrizen (position weight matrices, PWMs) etabliert.

In diesem Abschnitt verwenden wir ausschließlich das DNA-Alphabet $\Sigma := \{\texttt{A,C,G,T}\}$, formulieren die Definition jedoch allgemein.

4.1 Definition (PWM)**.** Eine Positions-Gewichts-Matrix (PWM) der Länge m über dem Alphabet Σ ist eine reellwertige $|\Sigma| \times m$-Matrix $S = (s_{c,j})_{c\in\Sigma, j\in\{0,\ldots,m-1\}}$. Jedem String $x \in \Sigma^m$ wird durch die PWM S ein Score zugeordnet, nämlich

$$\text{score}(x) := \sum_{j=0}^{m-1} S_{x[j],j}. \tag{4.1}$$

Man addiert also anhand der Symbole in x von links nach rechts die entsprechenden Score-Werte der Matrix.

4.2 Beispiel (PWM)**.** Sei die Reihenfolge der Zeilen durch `A,C,G,T` festgelegt und

$$S := \begin{pmatrix} -23 & 17 & -6 \\ -15 & -13 & -3 \\ -16 & 2 & -4 \\ 17 & -14 & 5 \end{pmatrix}.$$

Dann ist score(`TGT`) $= 17 + 2 + 5 = 24$. Die zu diesem Pattern am besten passende Sequenz (maximale Punktzahl) ist `TAT` mit Score 39. ♡

Die Matrizen für viele Transkriptionsfaktoren in verschiedenen Organismen sind bekannt und in öffentlichen oder kommerziellen Datenbanken wie JASPAR (`http://jaspar.genereg.net/`) oder TRANSFAC (`http://www.gene-regulation.com/pub/databases.html`) verfügbar. Die Matrizen werden anhand von beobachteten und experimentell verifizierten Bindestellen erstellt; auf die genaue Schätzmethode gehen wir in Abschnitt 4.3 ein. Zunächst betrachten wir aber das Pattern-Matching-Problem mit PWMs.

4.2 Pattern-Matching mit PWMs

Wir gehen davon aus, dass eine PWM S der Länge m, ein Text $T \in \Sigma^n$ und ein Score-Schwellenwert $t \in \mathbb{R}$ gegeben sind. Sei x_i das Fenster der Länge m, das im Text an Position i beginnt: $x_i := T[i \ldots i + m - 1]$. Gesucht sind alle Positionen i, so dass $\text{score}(x_i) \geq t$ ist.

Naive Verfahren. Der naive Algorithmus besteht darin, das Fenster x_i für $i = 0, \ldots, n-m$ über den Text zu schieben, in jedem Fenster $\text{score}(x_i)$ gemäß (4.1) zu berechnen und mit t zu vergleichen. Die Laufzeit ist offensichtlich $\mathcal{O}(mn)$.

Eine Alternative besteht darin, alle $|\Sigma|^m$ möglichen Wörter der Länge m aufzuzählen, ihren Score zu berechnen, und aus den Wörtern mit Score $\geq t$ einen Trie und daraus den Aho-Corasick-Automaten zu erstellen. Mit diesem kann der Text in $\mathcal{O}(n)$ Zeit durchsucht werden. Das ist nur sinnvoll, wenn $|\Sigma|^m \ll n$.

Lookahead Scoring. Besser als der naive Algorithmus ist, den Vergleich eines Fensters abzubrechen, sobald klar ist, dass entweder der Schwellenwert t nicht mehr erreicht werden kann (oder in jedem Fall überschritten wird). Dazu müssen wir wissen, welche Punktzahl im maximalen (minimalen) Fall noch erreicht werden kann. Wir berechnen also zu jeder Spalte j das Score-Maximum $M[j]$ über die verbleibenden Spalten:

$$M[j] := \sum_{k>j} \max_{c \in \Sigma} S_{c,k}.$$

Diese Liste der Maxima wird nur einmal für die PWM vorberechnet. (Analog kann man die entsprechende Liste der Minima berechnen – in der Praxis zeigt sich jedoch, dass dies kaum einen Geschwindigkeitsvorteil bietet und das Erkennen der Unmöglichkeiten, t noch zu erreichen, wesentlich effektiver ist.)

Beim Bearbeiten eines Fensters x kennen wir nach dem Lesen von $x[j]$ den partiellen Score

$$\text{score}_j(x) := \sum_{k=0}^{j} S_{x[k],k}.$$

Ist nun $\text{score}_j(x) + M[j] < t$, so kann t nicht mehr erreicht werden, und die Bearbeitung des Fensters wird erfolglos abgebrochen; ansonsten wird das nächste Zeichen evaluiert.

Abhängig von der Höhe des Schwellenwertes t lassen sich auf diese Weise viele Fenster nach wenigen Vergleichen abbrechen. Dieselbe Idee hilft bei einer Trie-Konstruktion, nur die Strings aufzuzählen, die tatsächlich den Trie bilden, statt aller 4^m Strings.

Permuted Lookahead Scoring. Lookahead Scoring ist vor allem dann effizient, wenn anhand der ersten Positionen eine Entscheidung getroffen werden kann, ob der Schwellenwert t noch erreichbar ist. Bei PWMs, die Motiven entsprechen, bei denen weit links viele mehrdeutige Zeichen stehen (zum Beispiel `ANNNNGTCGT`: In den `N`-Spalten wären alle Werte in der PWM gleich, zum Beispiel Null) hilft diese Strategie nicht viel. In welcher Reihenfolge wir aber die Spalten einer PWM (und die zugehörigen Textfensterpositionen) betrachten, spielt keine Rolle. Wir können also die Spalten vorab so umsortieren, dass „informative“ Spalten vorne stehen. Mit derselben Permutation muss dann in jedem Textfenster gesucht werden. Wie kann man eine gute Permutation π finden? Das Ziel ist, die erwartete Anzahl der verarbeiteten Textzeichen pro Fenster zu minimieren.

Sei $\text{score}_j^\pi(x) := \sum_{k=0}^{j} S_{x[\pi_k],\pi_k}$ der partielle Score bis einschließlich Position j nach Umsortieren mit der Permutation π, wenn das Fenster mit Inhalt x betrachtet wird.

Sei Y^π die Zufallsvariable, die nach Umsortierung mit der Permutation π die Anzahl der Textzeichen in einem Fenster zählt, die gelesen werden müssen, bis entweder die Entscheidung getroffen werden kann, dass der Schwellenwert t nicht erreicht wird, oder das Fenster vollständig abgearbeitet wurde. Für $0 \leq j < m-1$ ist man genau dann mit Spalte j fertig ($Y^\pi = j+1$), wenn $\text{score}_j^\pi(X) + M[j] < t$, aber noch $\text{score}_k^\pi(X) + M[k] \geq t$ für alle $k < j$. Dabei steht X für ein „zufälliges“ Textfenster der Länge m, also ein Textfenster, das der Hintergrundverteilung der Nukleotide des betrachteten Organismus folgt. Nach Spalte $m-1$ ist man in jedem Fall fertig; dann hat man m Zeichen gelesen ($Y^\pi = m$).

Nach Definition des Erwartungswertes ist also

$$\begin{aligned}\mathbb{E}[Y^\pi] &= \sum_{j=0}^{m-2} (j+1) \cdot \mathbb{P}(Y^\pi = j+1) + m \cdot \mathbb{P}(Y^\pi = m) \\ &= \sum_{j=0}^{m-2} (j+1) \cdot \mathbb{P}(\text{score}_k^\pi(X) + M[k] \geq t \text{ für } 0 \leq k < j \text{ und } \text{score}_j^\pi(X) + M[j] < t) \\ &\quad + m \cdot \mathbb{P}(\text{score}_k^\pi(X) + M[k] \geq t \text{ für alle } 0 \leq k < m-1).\end{aligned}$$

Prinzipiell können diese Wahrscheinlichkeiten und damit der Erwartungswert für jede Permutation π berechnet werden. Das ist jedoch insbesondere für lange TFBSen zu aufwändig.

Daher behilft man sich mit Heuristiken. Ein gutes Argument ist wie folgt: Aussagekräftig sind insbesondere Spalten, in denen Scores stark unterschiedlich sind (also das Maximum sehr viel größer ist als der zweitgrößte Wert oder als der Mittelwert der Spalte). Dies erlaubt eine schnelle (wenn auch eventuell nicht optimale) Anordnung der Spalten, die deutlich besser als das normale Lookahead-Scoring abschneidet.

Index-basierte Verfahren*. Muss man denselben sehr langen Text (zum Beispiel ein ganzes Genom) mit vielen verschiedenen PWMs durchsuchen, empfliehlt sich, einen Index (konkret: Ein Suffixarray `pos` samt `lcp`-Tabelle, sowie einer weiteren Hilfstabelle `skip` aufzubauen (siehe Skript "Algorithmen auf Sequenzen"). Auf diese Weise werden die Positionen des Textes nicht von links nach rechts, sondern lexikographisch durchsucht. Die partiellen Präfix-Scores müssen zwischen zwei aufeinanderfolgenden Positionen im Suffixarray, $\texttt{pos}[r]$ und $\texttt{pos}[r+1]$ nicht vollständig neu berechnet werden, denn das Präfix der Länge $\texttt{lcp}[r]$ haben diese Positionen ja gemeinsam. (Man muss also den Score für jedes Präfix bis ggf. zur Länge m speichern; dies sind aber nur wenige Werte. Bei Bedarf geht man dann zu der Länge $\texttt{lcp}[r]$ zurück.) Sobald klar ist, dass für ein bestimmtes Länge-ℓ Präfix der Schwellenwert t nicht mehr erreicht werden kann, kann man direkt alle lexikographisch folgenden Positionen, die dasselbe Länge-ℓ Präfix aufweisen, überspringen, bis man zu einer Stelle nicht größerem als dem aktuellen `lcp`-Wert kommt. Dies geschieht entweder durch lineares Durchscannen des `lcp`-Arrays oder durch Vorberechnen: Sei $\texttt{skip}[r]$ der kleinste Wert $r' \geq r$, so dass $\texttt{lcp}[r'] \leq \texttt{lcp}[r]$.

Beispiel: Wir haben gerade das Präfix `CGCCA` (erfolgreich) untersucht und gelangen nun am Index r zur lexikographisch nächsten Sequenz `CGCCC`, so dass $\texttt{lcp}[r] = 4$. Wir finden heraus, dass diese Sequenz den Score-Schwellenwert t nicht erreicht. Wir könnten also direkt zum Index r' springen, bei dem die Sequenzen mit `CGCCG` beginnen (wenn es solche gibt). An einer solchen Stelle gilt offenbar $\texttt{lcp}[r'] \leq 4$, während die `lcp`-Werte dazwischen alle mindestens 5 betragen müssen. Entweder haben wir diesen Index r' in $\texttt{skip}[r]$ vorberechnet, oder wir finden ihn durch cache-effizientes Durchscannen von `lcp`.

4.3 Schätzen von PWMs

Eine PWM wird aus (experimentell validierten) Beispielsequenzen einer TFBS erstellt. Die erste vereinfachende Annahme ist, dass man alle Positionen unabhängig voneinander betrachten kann. An jeder Position j wird also gezählt, wie oft jedes Zeichen $c \in \Sigma$ beobachtet

wurde. Dies liefert eine *Zählmatrix* ($N_{c,j}$), in der die „horizontalen“ Abhängigkeiten zwischen den einzelnen Positionen verlorengegangen sind, d.h. es ist nicht möglich, die Beispielsequenzen aus der Matrix zu rekonstruieren.

Teilt man die Zählmatrix spaltenweise durch die Gesamtzahl der Beobachtungen in jeder Spalte (“Normalisierung”), so erhält man eine *Wahrscheinlichkeitsmatrix* (auch *Profil*) ($P_{c,j}$). Hier summieren sich die Spalten zum Wert 1. Bei wenigen Beispielsequenzen kommt es vor, dass an manchen Positionen manche Nukleotide gar nicht beobachtet wurden; dort ist $N_{c,j} = P_{c,j} = 0$. Nun muss es aber nicht unmöglich sein, dass man in Zukunft eine entsprechende Beobachtung macht. Eine Wahrscheinlichkeit von 0 bedeutet jedoch ein unmögliches Ereignis. Daher werden vor der Division zu allen Einträgen sogenannte *Pseudocounts* hinzugezählt (z.B. jeweils 1). Dies lässt sich auch lerntheoretisch und mit Methoden der Bayes'schen Statistik begründen. Man spricht dabei von *Regularisierung*. Im Ergebnis enthält das Profil nun kleine Wahrscheinlichkeiten für nicht beobachtete Ereignisse und Wahrscheinlichkeiten, die in etwa proportional zur beobachteten Häufigkeit sind, für beobachtete Ereignisse.

Der Sequenzbereich, in dem man sucht, wird normalerweise eine bestimmte Verteilung von Nukleotiden aufweisen, die nicht die Gleichverteilung ist. Angenommen, das Profil enthält an manchen Positionen mit großer Wahrscheinlichkeit ein `G`. Wenn auch die durchsuchte DNA-Region `GC`-reich ist, ist es nicht so überraschend, ein `G` an dieser Stelle zu sehen, wie in einer `AT`-reichen Region. Die Bewertung der Übereinstimmung eines Fensters x sollte daher immer relativ zum globalen „Hintergrund“ erfolgen. Wir gehen also davon aus, dass ein Hintergrundmodell q gegeben ist, das jedem Symbol eine bestimmte Wahrscheinlichkeit zuordnet. Eine sehr `GC`-reiche Region könnte dann (in der Reihenfolge `ACGT`) durch $q = (0.2, 0.3, 0.3, 0.2)$ beschrieben werden.

Die PWM erhält man nun als *log-odds* Matrix zwischen Profil und Hintergrundmodell, d.h. man betrachtet an jeder Stelle j das Verhältnis zwischen den Wahrscheinlichkeiten für Symbol c im Profil und im Hintergrundmodell, also $P_{c,j}/q_c$. Ist dieses Verhältnis größer als 1, dann steht das Zeichen an Position j für eine gute Übereinstimmung zur PWM. Äquivalent dazu betrachtet man den Logarithmus des Verhältnisses im Vergleich zu 0. Damit ist die PWM $S = (S_{c,j})$ durch

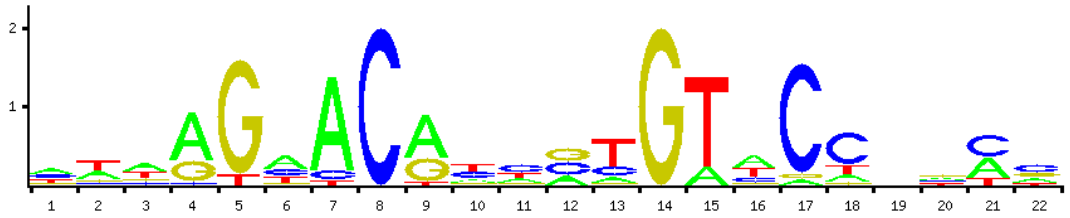
$$S_{c,j} := \log \frac{P_{c,j}}{q_c}$$

gegeben. Die Addition der Scores in einem Fenster entspricht einer Multiplikation der Wahrscheinlichkeiten über alle Positionen (Unabhängigkeitsannahme).

4.4 Sequenzlogos als Visualisierung von PWMs

Eine Matrix von Scores ist schlecht zu lesen. Man möchte aber natürlich wissen, wie Sequenzen, die einen hohen Score erreichen, typischerweise aussehen. Eine Möglichkeit besteht in der Angabe, eines *IUPAC-Konsensus-Strings*: Man wählt in jeder Spalte das Nukleotid mit höchstem Score aus, oder auch das IUPAC-Symbol für die 2/3/4 Nukleotide mit höchsten Scores, wenn die Unterschiede nicht sehr groß sind.

Eine ansprechendere Visualisierung gelingt durch sogenannte Sequenzlogos. Jede Position wird durch einen Turm oder Stapel aller Nukleotide dargestellt. Die Gesamthöhe des Turms




```
A  [ 9  9 11 16  0 12 21  0 15  4  5  6  3  0  4 11  1  3  6  6 10  5 ]
C  [ 7  2  3  1  0  6  2 24  0  9 11  9  5  0  0  5 22 16  7  5 11 11 ]
G  [ 2  3  3  7 22  1  0  0  8  2  2  9  1 24  0  1  1  1  5  9  0  6 ]
T  [ 6 10  7  0  2  5  1  0  1  9  6  0 15  0 20  7  0  4  6  4  3  2 ]
```

Abbildung 4.1: Oben: Sequenzlogo des Androgen-Transkriptionsfaktors der Ratte aus der JASPAR-Datenbank (`http://jaspar.genereg.net/cgi-bin/jaspar_db.pl?ID=MA0007.1&rm=present&collection=CORE`). Die x-Achse ist die Position (0 bis 21, dargestellt als 1 bis 22). Die y-Achse repräsentiert die Höhe in Bits. Unten: zugehörige Zählmatrix.

ist proportional zur mit den Profilwahrscheinlichkeiten gewichteten Summe aller Scores in dieser Spalte. Diese beträgt

$$h_j = \sum_c P_{c,j} S_{c,j} = \sum_c P_{c,j} \log \frac{P_{c,j}}{q_c}.$$

Dieser Ausdruck ist in der Informationstheorie auch bekannt als die *relative Entropie* zwischen der Verteilung $P_{\cdot,j}$ und der Verteilung q. Man kann beweisen, dass sie immer nichtnegativ ist.

Für die Visualisierung wird q oft einfach als Gleichverteilung auf dem DNA-Alphabet angenommen, so dass $q_c \equiv 1/4$. Nimmt man dann noch Logarithmen zur Basis 2 (dabei spricht man dann von Bits), so ist

$$h_j = \sum_c P_{c,j} \log_2 \frac{P_{c,j}}{1/4} = \sum_c P_{c,j}(2 + \log_2 P_{c,j}) = 2 + \sum_c P_{c,j} \log_2 P_{c,j}.$$

Da die Summe immer negativ ist, folgt $0 \leq h_j \leq 2$ [Bits].

Innerhalb dieser Gesamthöhe erhält dann jedes Symbol Platz proportional zu $P_{c,j}$. Die Höhe von Symbol c an Position j in der Visualisierung ist also $H_{c,j} = h_j \cdot P_{c,j}$. Ein Beispiel ist in Abbildung 4.1 gezeigt.

4.5 Wahl eines Schwellenwerts

Beim Pattern-Matching mit einer PWM muss man sich für einen Score-Schwellenwert t entscheiden. Je größer t gewählt wird, desto weniger Strings erreichen diesen Schwellenwert,

und die Suche wird spezifischer (und bei permuted lookahead scoring auch schneller). Wie aber sollte t gewählt werden?

Es gibt mehrere Kriterien, die sinnvoll sind, die sich aber teilweise widersprechen.

- Alle der in die PWM eingegangenen Beispielsequenzen sollten gefunden werden.
- Die Wahrscheinlichkeit, dass ein zufälliges Textfenster ein Match ist, sollte klein sein, vielleicht 0.01. Damit findet man dann (im Mittel) alle 100 Positionen einen „zufälligen“ Match.
- Die Wahrscheinlichkeit, dass ein nach dem Profil erstellter String ein Match ist, sollte groß sein, vielleicht 0.95.

Um t zu wie oben vorgegebenen Wahrscheinlichkeiten passend zu wählen, muss man in der Lage sein, die Score-Verteilung einer PWM unter verschiedenen Modellen (unter dem Hintergrundmodell q und unter dem PWM-Modell P selbst) zu berechnen. Dies ist effizient möglich, zum Beispiel mit den im nächsten Kapitel vorgestellten probabilistischen arithmetischen Automaten.

KAPITEL 5

Probabilistische Arithmetische Automaten

In diesem Kapitel entwickeln wir einen mathematischen Rahmen (probabilistische arithmetische Automaten), der es erlaubt, Berechnungen auf allgemeinen Textmodellen anzustellen.

5.1 Motivation: Nichtvorkommen einer Zeichenkette

Als eine Anwendung betrachten wir folgende Frage: Gegeben sind ein Textmodell, ein Muster und ein beobachteter Text einer bestimmten Länge. Wir sehen, dass das Muster k-mal vorkommt und fragen uns, ob das überraschend viele oder wenig Vorkommen sind. Die Frage ist also, wie typische Texte aus dem Textmodell im Vergleich zum beobachteten Text bezüglich der Anzahl der Vorkommen abschneiden, oder mit anderen Worten: Wie ist die Verteilung der Anzahl der "Treffer" des Musters in Texten aus dem Textmodell?

Wir wenden uns jetzt der Berechnung der Verteilung der Anzahl der Treffer zu. Als Spezialfall betrachten wir die Frage nach der Wahrscheinlichkeit, dass überhaupt *kein* Treffer im Text vorkommt.

5.1 Beispiel (Verteilung der Treffer eines Musters)**.** Sei $\Sigma = \{\texttt{O},\texttt{M},\texttt{A}\}$; betrachte die Motive $M_1 = \texttt{OMAMA}$ und $M_2 = \texttt{MAOAM}$ und z.B. die Textlänge $n = 9$. Viele finden es verblüffend, dass die Wahrscheinlichkeit, dass M_1 nicht vorkommt, verschieden ist von der Wahrscheinlichkeit, dass M_2 nicht vorkommt. Tatsächlich verteilen sich 0, 1 und 2 Vorkommen auf die $3^9 = 19683$ Strings wie folgt.

	Anzahl			Anteil		
Motiv	0 Treffer	1 Treffer	2 Treffer	0 Treffer	1 Treffer	2 Treffer
OMAMA	19278	405	0	0.97942	0.02058	0.00000
MAOAM	19279	403	1	0.97947	0.02047	0.00005

Die 405 Strings für das nichtüberlappende `OMAMA` ergeben sich aus den 5 verschiedenen Startpositionen, an denen das Wort beginnen kann und die freie Wahl der verbleibenden 4 Zeichen im String; dabei wird kein String doppelt gezählt, da ein Vorkommen von `OMAMA` im String ein weiteres ausschließt. Es ist in der Tat $5 \cdot 4^3 = 405$. ♡

Aufgabe 5.1. Man könnte vermuten, dass die Verteilung bei `MAOAM`, wenn es schon einen einzelnen String mit 2 Treffern gibt, so aussieht: $(19278, 404, 1)$, d.h., dass zumindest die Anzahl der Strings mit keinem Treffer gegenüber `OMAMA` unverändert ist. Benutze ein einfaches Argument, um zu erklären, warum das nicht der Fall sein kann.

Aufgabe 5.2. Konstruiere ein Minimalbeispiel (binäres Alphabet, möglichst kurze Wörter, kurzer Text) für dasselbe Phänomen wie in Beispiel 5.1.

Wir beschreiben jetzt, ausgehend von einem deterministischen endlichen Automaten, eine Konstruktion, mit der sich nicht nur die Anzahlen (bzw. Anteile im M00-Modell), sondern allgemein die Trefferverteilung eines Musters in einem allgemeinesn Textmodell berechnen lässt: probabilistische arithmetische Automaten.

Die Idee ist, salopp formuliert, nicht in einem festen String nach einem Muster zu suchen, sondern in einem Zufallsstring. Wir verwenden also einen Automaten, der nicht in einem konkreten Zustand ist, sondern in jedem Zustand mit einer bestimmten Wahrscheinlichkeit. Wir werden dann diese Wahrscheinlichkeitsverteilung Schritt für Schritt aktualisieren. Wir wissen also zu jedem Zeitpunkt, mit welcher Wahrscheinlichkeit wir in welchem Zustand sind.

5.2 Definition eines PAAs

Wir definieren *probabilistische arithmetische Automaten* (PAAs), um Folgen von Operationen ("Berechnungen") mit probabilisitschen Operanden zu formalisieren.

5.2 Definition (Probabilistischer arithmetischer Automat)**.** Ein *probabilistischer arithmetischer Automat* ist ein 8-Tupel

$$\mathcal{P} = \big(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu = (\mu_q)_{q\in\mathcal{Q}}, \theta = (\theta_q)_{q\in\mathcal{Q}}\big)\,;$$

- $\mathcal{Q}$ ist eine endliche *Zustandsmenge*,
- $q_0 \in \mathcal{Q}$ heißt *Startzustand*,
- $T : \mathcal{Q} \times \mathcal{Q} \to [0,1]$ ist eine Übergangsmatrix mit $\sum_{q'\in\mathcal{Q}} T(q,q') = 1$ für alle $q \in \mathcal{Q}$, i.e. $\big(T(q,q')\big)_{q,q'\in\mathcal{Q}}$ ist eine stochastische Matrix,
- $\mathcal{V}$ heißt *Wertemenge*,
- $v_0 \in \mathcal{V}$ ist der *Startwert*,
- $\mathcal{E}$ ist eineendliche Menge, die *Emissionsmenge*,
- $\mu_q : \mathcal{E} \to [0,1]$ ist die Emissionsverteilung im Zustand q; eine solche ist für alle $q \in \mathcal{Q}$ gegeben,

- $\theta_q : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$ definiert die Operation im Zustand q; eine solche ist für alle $q \in \mathcal{Q}$ gegeben.

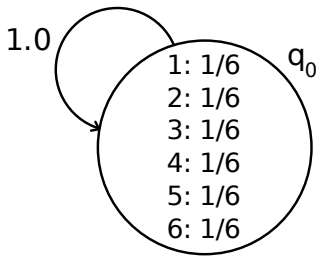
Wir versehen die Definition mit der folgenden Semantik: Der Automat befindet sich zu jedem Zeitpunkt in genau einem Zustand $q \in \mathcal{Q}$, beginnend im Startzustand q_0. Die Übergänge sind probabilistisch gemäß der Ubergangsmatrix T, so dass $T(q, q')$ die Wahrscheinlichkeit ist, in den Zustand q' überzugehen, wenn man sich gerade im Zustand q befindet. Damit bildet $(\mathcal{Q}, q_0, T)$ eine Markovkette.

Während der Ubergänge führt der PAA eine Abfolge von Operationen auf der Wertemenge aus, beginnend mit Wert v_0. Nach jedem Zustandsübergang erzeugt der betretene Zustand q eine Emission aus der Menge $\mathcal{E}$ gemäß der Verteilung μ_q. Der aktuelle Wert und die erzeugte Emission werden dann durch die zustandsspezifische Operation θ_q zu einem neuen Wert aus $\mathcal{V}$ verrechnet.

Die Markovkette $(\mathcal{Q}, T, \boldsymbol{\delta}_{q_0})$ bildet zusammen mit der Emissionsmenge $\mathcal{E}$ und den Emissionsverteilungen $\mu = (\mu_q)_{q \in \mathcal{Q}}$ ein hidden Markov model (HMM). Bei HMMs untersuchen wir jedoch in der Regel die Folge der Emissionen mit dem Ziel auf die Zustandssequenz zu schließen, während uns bei PAAs die aus den Emissionen berechneten Werte (und deren Verteilung) interessieren.

PAAs erlauben so eine natürliche Modellierung verschiedener Anwendungsfragestellungen. Tatsächlich sind PAAs aber nicht mächtiger als einfache Markovketten: Aus theoretischer Sicht kann man jeden PAA als Markovkette auf dem Zustandsraum $\mathcal{Q} \times \mathcal{V}$ auffassen. Dies sollte man jedoch nicht immer tun, da die Modellierung mit PAAs deutlich intuitiver sein kann.

5.3 Beispiel (Einfacher PAA)**.** Wir betrachten einen sehr einfachen Automaten, der uns Auskunft geben kann, wie groß die Summe der Augen nach einer bestimmten Anzahl an Würfelwürfen ist.



Formal definieren wir unseren PAA durch. $Q = \{q_0\}$, $T = (1)$, $E = \{1, 2, 3, 4, 5, 6\}$, $e_{q_0} \equiv 1/6$, $N = \mathbb{N}$, $n_0 = 0$, $\theta_{q_0} : (v, e) \mapsto v + e$. ♡

Wir geben nun formal die stochastischen Prozesse an, die durch die Semantik eines PAAs entstehen.

5.4 Definition (Stochastische Prozesse eines PAAs)**.** Sei $\mathcal{P} = (\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ ein PAA. Mit $(Q_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$ bezeichnen wir seinen *Zustandsprozess*. Diesen definieren wir als Markovkette mit $Q_0^{\mathcal{P}} \equiv q_0$ und

$$\begin{aligned} &\mathbb{P}\left(Q_{t+1}^{\mathcal{P}} = q_{t+1} \mid Q_t^{\mathcal{P}} = q_t, \ldots, Q_0^{\mathcal{P}} = q_0\right) \\ :=\ &\mathbb{P}\left(Q_{t+1}^{\mathcal{P}} = q_{t+1} \mid Q_t^{\mathcal{P}} = q_t\right) \\ :=\ &T(q_t, q_{t+1}) \end{aligned} \tag{5.1}$$

für alle $q_0, \ldots q_{t+1} \in \mathcal{Q}$.

Den *Emissionsprozess* $(E_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$ definieren wir durch

$$\begin{aligned} &\mathbb{P}(E_t^{\mathcal{P}} = e \mid Q_0^{\mathcal{P}} = q_0, \ldots, Q_t^{\mathcal{P}} = q_t, E_0^{\mathcal{P}} = e_0, \ldots, E_{t-1}^{\mathcal{P}} = e_{t-1}) \\ :=\ &\mathbb{P}\left(E_t^{\mathcal{P}} = e \mid Q_t^{\mathcal{P}} = q\right) \\ :=\ &\mu_q(e), \end{aligned} \tag{5.2}$$

d.h. die aktuelle Emission hänge nur vom aktuellen Zustand ab.

Wir benutzen $(Q_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$ und $(E_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$, um den *Werteprozess* $(V_t^{\mathcal{P}})_{t\in\mathbb{N}_0}$, der aus den Operationen resultiert, zu definieren:

$$V_0^{\mathcal{P}} :\equiv v_0 \quad \text{und} \quad V_t^{\mathcal{P}} := \theta_{Q_t^{\mathcal{P}}}\left(V_{t-1}^{\mathcal{P}}, E_t^{\mathcal{P}}\right). \tag{5.3}$$

Wenn aus dem Kontext klar ist, welcher PAA gemeint ist, lassen wir das hochgestellte $\mathcal{P}$ weg und schreiben einfach jeweils $(Q_t)_{t\in\mathbb{N}_0}$, $(V_t)_{t\in\mathbb{N}_0}$ bzw. $(E_t)_{t\in\mathbb{N}_0}$.

5.3 Berechnung der Zustands-Werte-Verteilung von PAAs

Das Ziel der Modellierung einer Frage mit PAAs ist, die Werteverteilung nach einer bestimmten Anzahl n von Schritten berechnen zu können. Mit anderen Worten, gesucht ist die Verteilung $\mathcal{L}(V_n)$ der Zufallsvariablen V_n für gegebenes n. Hierfür beschreiben wir zwei Algorithmen.

Die Idee ist, zunächst die gemeinsame Verteilung $\mathcal{L}(Q_n, V_n)$ von Zuständen und Werten zu berechnen, um daraus durch Marginalisierung die Verteilung der Werte zu gewinnen:

$$\mathbb{P}(V_n = v) = \sum_{q\in\mathcal{Q}} \mathbb{P}(Q_n = q, V_n = v). \tag{5.4}$$

Zur Abkürzung definieren wir $f_t(q, v) := \mathbb{P}(Q_t = q, V_t = v)$ für $t \in \mathbb{N}_0$, $q \in \mathcal{Q}$, $v \in \mathcal{V}$.

Ist $\mathcal{V}$ nicht endlich, ergibt sich eine Komplikation bei der Beschreibung der Algorithmen und ihrer Laufzeit. Jedoch ist für jedes endliche t der Wertebereich von V_t endlich, da er eine Funktion der (endlich vielen) möglichen Zustände und Emissionen bis zur Zeit t ist. Wir definieren $\mathcal{V}_t := \operatorname{range} V_t$ und $\vartheta_n := \max_{0\le t\le n} |\mathcal{V}_t|$. Es ist $\vartheta_n \le (|\mathcal{Q}| \cdot |\mathcal{E}|)^n$. Daher lassen sich alle Berechnungen auf endlichen (jedoch für wachsende t schlimmstenfalls exponentiell wachsenden) Mengen anstellen. In vielen Anwendungen wächst ϑ_n nur polynomiell (oder auch nur linear) mit n. In the following, we shall understand $\mathcal{V}$ as the appropriate union of $\mathcal{V}_t$ sets. Running times of algorithms are given in terms of ϑ_n.

5.3.1 Grundlegende Rekurrenz

Die Grundlegende Rekurrenz zur Berechnung der Zustands-Werte-Verteilung $f_n = \mathcal{L}(Q_n, V_n)$ folgt direkt aus den Definitionen 5.2 and 5.4.

5.5 Lemma (Rekurrenz der Zustands-Werte-Verteilung)**.** Für einen gegebenen PAA kann die Zustands-Werte-Verteilung wie folgt berechnet werden:

$$f_0(q,v) = \begin{cases} 1 & \text{wenn } q = q_0 \text{ und } v = v_0\,, \\ 0 & \text{sonst}\,, \end{cases} \tag{5.5}$$

$$f_{t+1}(q,v) = \sum_{q' \in \mathcal{Q}} \sum_{(v',e) \in \theta_q^{-1}(v)} f_t(q',v') \cdot T(q',q) \cdot \mu_q(e)\,, \tag{5.6}$$

wobei $\theta_q^{-1}(v)$ die Urbildmenge von v unter θ_q ist.

Beweis. Die Initialisierung (5.5) folgt direkt aus (5.1) und (5.3). Wir verifizieren also die Rekurrenz (5.6):

$$\begin{aligned}
& f_{t+1}(q,v) \\
=& \mathbb{P}\left(Q_{t+1} = q, V_{t+1} = v\right) \\
=& \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} \sum_{e \in \mathcal{E}} \mathbb{P}(Q_{t+1} = q, V_{t+1} = v, E_{t+1} = e, Q_t = q', V_t = v') \\
=& \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} \sum_{e \in \mathcal{E}} f_t(q',v') \cdot \mathbb{P}(Q_{t+1} = q, V_{t+1} = v, E_{t+1} = e \mid Q_t = q', V_t = v') \\
=& \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} \sum_{e \in \mathcal{E}} [\![\theta_q(v',e) = v]\!] \cdot f_t(q',v') \cdot \mathbb{P}\left(Q_{t+1} = q, E_{t+1} = e \mid Q_t = q', V_t = v'\right) \\
=& \sum_{q' \in \mathcal{Q}} \sum_{(v',e) \in \theta_q^{-1}(v)} f_t(q',v') \cdot \underbrace{\mathbb{P}\left(Q_{t+1} = q, E_{t+1} = e \mid Q_t = q', V_t = v'\right)}_{(*)}\,.
\end{aligned}$$

Wir werten nun weiter den Ausruck $(*)$ aus:

$$\begin{aligned}
(*) =& \mathbb{P}\left(Q_{t+1} = q, E_{t+1} = e \mid Q_t = q', V_t = v'\right) \\
=& \underbrace{\mathbb{P}(E_{t+1} = e \mid Q_t = q', Q_{t+1} = q, V_t = v')}_{\overset{(i)}{=} \mu_q(e)} \cdot \underbrace{\mathbb{P}\left(Q_{t+1} = q \mid Q_t = q', V_t = v'\right)}_{\overset{(ii)}{=} T(q',q)}\,;
\end{aligned}$$

dabei folgt (i) aus (5.2) und (5.3), und (ii) aus der Tatsache, dass $(Q_t)_{t \in \mathbb{N}_0}$ eine Markovkette ist. □

Zur Berechnung beginnt man also mit der (trivialen) Verteilung f_0 und wendet die Rekurrenz (5.6) n-mal an, bis die gewünschte Verteilung berechnet ist.

Die Rekurrenz ist (wie üblich) in der *pull*-Formulierung angegeben: Bei der Berechnung des aktuellen Wertes in f_{t+1} "zieht" man die benötigten vorherigen Werte aus f_t heran. Unschön hierbei ist, dass man die Urbilder $\theta_q^{-1}(v)$ kennen (also ggf. vorberechnen) muss, um die Laufzeit effizient zu halten. Eine algorithmische *push*-Formulierung hat den Vorteil, dass dies nicht notwendig ist und die Laufzeit einfacher abgeschätzt werden kann: Wir iterieren über alle f_t-Werte und "schieben" diese an die richtige(n) Stelle(n) von f_{t+1}. Algorithmus 1 zeigt eine effiziente Implementierung.

Algorithm 1 PAADIST

Input: $f_0 = \mathcal{L}(Q_0, V_0)$, $n \in \mathbb{N}_0$, Platz für zwei Tabellen der Größe $|\mathcal{Q}| \times \vartheta_n$
Output: $f_n = \mathcal{L}(Q_n, V_n)$

```
for t = 1 to n do
    initialisiere f_t(q, v) :≡ 0 für alle q ∈ Q, v ∈ V_t
    for all q ∈ Q und v ∈ V_{t-1} do
        for all q' ∈ Q und e ∈ E do
            v' ← θ_{q'}(v, e)
            f_t(q', v') ← f_t(q', v') + f_{t-1}(q, v) · T(q, q') · μ_{q'}(e)
return f_n
```

5.6 Lemma (Laufzeit)**.** Gegeben sei ein PAA $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$. Dann kann die Verteilung der Werte $\mathcal{L}(V_n)$ oder die Zustands-Werte-Verteilung $\mathcal{L}(Q_n, V_n)$ in $\mathcal{O}(n \cdot |\mathcal{Q}|^2 \cdot \vartheta_n \cdot |\mathcal{E}|)$ Zeit und $\mathcal{O}(|\mathcal{Q}| \cdot \vartheta_n)$ Platz berechnet werden.

Beweis. Der Beweis folgt direkt aus Algorithmus 1 durch Zählen der Operationen. (Insbesondere wird bei der *push*-Formulierung eher klar, dass ϑ_n linear und nicht quadratisch in die Laufzeit eingeht, was an Hand der Rekurrenz (5.6) erst sorgfältig zu prüfen wäre.) □

5.3.2 Verdopplungs-Technik

Wenn n sehr groß ist, wird die grundlegende Rekurrenz langsam, da ihre Laufzeit linear von n abhängt. Wir stellen hier eine Alternative vor, die unter bestimmten Bedingungen für große n schneller sein kann.

Wir betrachten die bedingte Wahrscheinlichkeit

$$\begin{aligned} &U^{(t)}(q_1, q_2, v_1, v_2) \\ :=\ &\mathbb{P}\big(Q_{t_0+t} = q_2, V_{t_0+t} = v_2 \,\big|\, Q_{t_0} = q_1, V_{t_0} = v_1\big), \end{aligned}$$

innherhalb von t Zeitschritten von (q_1, v_1) zu (q_2, v_2) überzugehen. Beachte, dass $U^{(t)}$ nicht vom Startzeitpunkt t_0 abhängt, da sich weder die Übergangs- noch die Emissionswahrscheinlichkeiten im Lauf der Zeit ändern (diese Eigenschaft nennt man *Homogenität*). Hat man $U^{(n)}$ berechnet, dann kann man die gewünschte Zustands-Werte-Verteilung $\mathcal{L}(Q_n, V_n)$ einfach ablesen:

$$\mathbb{P}(Q_n = q, V_n = v) = U^{(n)}(q_0, q, v_0, v).$$

Das folgende Lemma zeigt, wie man $U^{(t)}$ in nur logarithmisch vielen Zeitschritten berechnen kann.

5.7 Lemma (Verdopplungs-Lemma)**.** Sei $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ ein PAA, und seien $(Q_t)_{t\in\mathbb{N}_0}$ und $(V_t)_{t\in\mathbb{N}_0}$ jeweils sein Zustands- und Werteprozess. Dann ist

$$U^{(1)}(q_1, q_2, v_1, v_2) = T(q_1, q_2) \cdot \sum_{\substack{e\in\mathcal{E}:\\ \theta_{q_2}(v_1,e)=v_2}} \mu_{q_2}(e), \tag{5.7}$$

sowie, für alle $t_1 \in \mathbb{N}_0$ und $t_2 \in \mathbb{N}_0$,

$$\begin{aligned} & U^{(t_1+t_2)}(q_1, q_2, v_1, v_2) \\ = & \sum_{q' \in \mathcal{Q}} \sum_{v' \in \mathcal{V}} U^{(t_1)}(q_1, q', v_1, v') \cdot U^{(t_2)}(q', q_2, v', v_2). \end{aligned} \tag{5.8}$$

Es folgt (durch mehrfache Verdopplung von t), dass die Werteverteilung $\mathcal{L}(V_n)$ in $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot \vartheta_n^3)$ Zeit und mit $\mathcal{O}(|\mathcal{Q}|^2 \cdot \vartheta_n^2)$ Platz berechnet werden kann. Diese Technik ist asymptotisch schneller, wenn ϑ_n in $o(\sqrt{n/\log n})$ liegt (also nur sehr schwach mit n wächst) und $\mathcal{Q}$ und $\mathcal{E}$ konstant groß sind.

Beweis. Die Initialisierung (5.7) folgt direkt aus Definition 5.4. Zum Beweis von Gleichung (5.8) erinnern wir daran, dass ein PAA auch einfach als Markovkette auf $\mathcal{Q} \times \mathcal{V}$ aufgefasst werden kann. Damit ist (5.8) einfach die bekannte Chapman-Kolmogorov-Gleichung für Markovketten, die besagt, dass man die Übergangsmatrix für $t_1 + t_2$ Schritte durch Multiplikation der jeweiligen Übergangsmatrizen für t_1 und t_2 Schritte erhält.

Zur Laufzeit: Die Berechnung von $U^{(t_1+t_2)}$ aus $U^{(t_1)}$ und $U^{(t_2)}$ kostet $\mathcal{O}(|\mathcal{Q}|^3 \cdot \vartheta_n^3)$ Zeit, wie man aus (5.8) erkennt. Man kann in einem solchen Schritt $U^{(2t)}$ aus $U^{(t)}$ berechnen. In $\lceil \log(n) \rceil$ Schritten erhält man also alle $U^{(2^b)}$ für $0 \leq b \leq \lceil \log(n) \rceil$ und kann diese zu $U^{(n)}$ kombinieren. □

5.4 Wartezeiten

Neben der Berechnung der Zustands-Werte-Verteilung nach einer festen Anzahl von Schritten ist es für viele Anwendungen wichtig, die Verteilung der Anzahl der Schritte zu bestimmen, bis ein bestimmtes Ereignis eintritt, etwa bis ein bestimmter Zustand oder Wert erreicht wird. Bei den Anwendungen in den folgenden Abschnitten wird die Bedeutung solcher *Wartezeitprobleme* deutlich.

5.8 Definition (Wartezeitprobleme)**.** Wir unterscheide Wartezeitprobleme für Werte und Zustände und eine Kombination von beidem.

1. Sei $\mathcal{T} \subset \mathcal{V}$ eine *Zielmenge* von Werten. Dann ist die Zufallsvariable $W_{\mathcal{T}}$ der Wartezeit für diese Zielmenge definiert als $W_{\mathcal{T}} := \min\{t \in \mathbb{N}_0 \mid V_t \in \mathcal{T}\}$, wenn diese Menge nicht leer ist, und ansonsten als unendlich.

2. Sei $\mathcal{S} \subset \mathcal{Q}$ eine Zielmenge von Zuständen. Dann ist die Zufallsvariable $W_{\mathcal{S}}$ der Wartezeit für diese Zielmenge definiert als $W_{\mathcal{S}} := \min\{t \in \mathbb{N}_0 \mid Q_t \in \mathcal{S}\}$, wenn diese Menge nicht leer ist, und ansonsten als unendlich.

3. Sei $\mathcal{R} \subset \mathcal{Q} \times \mathcal{V}$ eine Zielmenge von Zustands-Werte-Paaren. Dann ist die Zufallsvariable $W_{\mathcal{R}}$ der Wartezeit für diese Zielmenge definiert als $W_{\mathcal{R}} := \min\{t \in \mathbb{N}_0 \mid (Q_t, V_t) \in \mathcal{R}$, wenn diese Menge nicht leer ist, und ansonsten als unendlich.

Wir betrachten im folgenden zunächst den Fall einer Zielmenge für Werte und die Wartezeit $W_{\mathcal{T}}$.

Während $\mathbb{P}(W_{\mathcal{T}} \geq t)$ für alle $t \in \mathbb{N}$ echt größer als null sein kann, sind wir häufig nur an der Verteilung bis zu einer vorgegebenen Maximaldauer n interessiert. Dadurch wird das Problem wieder endlich. Natürlich kennen wir dann nicht die exakten Werte von $\mathcal{L}(W_{\mathcal{T}})(t) = \mathbb{P}(W_{\mathcal{T}} = t)$ für $t > n$, aber ihre Gesamtwahrscheinlichkeit $\mathbb{P}(W_{\mathcal{T}} > n)$ ist bekannt, und n kann so gewählt werden, dass diese Gesamtwahrscheinlichkeit vernachlässigbar klein ist.

Man kann das Problem der Berechnung der Verteilung der Wartezeit für eine Wertemenge $\mathcal{T}$ auf die im vorigen Abschnitt beschriebenen Berechnungen reduzieren, wenn man einen leicht modifizierten PAA $\mathcal{P}'$ definiert. Dieser erhält die neue Wertemenge $\mathcal{V}' := (\mathcal{V} \setminus \mathcal{T}) \cup \{\bullet, \circ\}$, wobei $\bullet, \circ \notin \mathcal{V}$ zwei neue besondere Werte sind und neue Operationen

$$\theta'_q(v,e) := \begin{cases} \theta_q(v,e) & \text{wenn } v \notin \{\bullet,\circ\} \text{ und } \theta_q(v,e) \notin \mathcal{T}, \\ \bullet & \text{wenn } v \notin \{\bullet,\circ\} \text{ und } \theta_q(v,e) \in \mathcal{T}, \\ \circ & \text{wenn } v \in \{\bullet,\circ\} \end{cases}$$

für alle $q \in \mathcal{Q}$. Sobald also $\mathcal{P}$ einen Wert aus $\mathcal{T}$ annehmen würde, nimmt $\mathcal{P}'$ den Wert $\bullet$ an und allen Schritt danach den Wert $\circ$.

Sei V'_t der Werteprozess des modifizierten PAAs $\mathcal{P}'$. Nun kann die Wahrscheinlichkeit, dass die Wartezeit genau t Schritte beträgt, durch

$$\mathbb{P}(W_{\mathcal{T}} = t) = \mathbb{P}(V'_t = \bullet)$$

ausgedrückt werden und die Berechnung verläuft für alle t wie in den Lemmas 5.6 and 5.7 beschrieben, je nach Algorithmus. Daraus erhält man auch die Laufzeiten.

5.9 Lemma (Laufzeiten)**.** Sei $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ ein PAA und $\mathcal{T} \subset \mathcal{V}$. Dann können die Wahrscheinlichkeiten $\mathcal{L}(W_{\mathcal{T}})(0), \ldots, \mathcal{L}(W_{\mathcal{T}})(n)$ in $\mathcal{O}\big(n \cdot |\mathcal{Q}|^2 \cdot |\mathcal{E}| \cdot (\vartheta_n - |\mathcal{T}|)\big)$ Zeit und mit $\mathcal{O}\big(|\mathcal{Q}| \cdot (\vartheta_n - |\mathcal{T}|)\big)$ Platz berechnet werden. Alternativ kann dies (mit der Verdopplungs-Technik) in $\mathcal{O}\big(\log n \cdot |\mathcal{Q}|^3 \cdot (\vartheta_n - |\mathcal{T}|)^3\big)$ Zeit und mit $\mathcal{O}\big(|\mathcal{Q}|^2 \cdot (\vartheta_n - |\mathcal{T}|)^2\big)$ Platz geschehen.

Das Warten auf einen Zustand aus einer Zielmenge $\mathcal{S} \subset \mathcal{Q}$ betrifft nur die Markovkette $(\mathcal{Q}, T, q_0)$ und ist oft beschrieben worden Reinert et al. (2000); Brémaud (1999); wir fassen hier die Resultate zusammen.

Man geht genau so vor wie bei dem Warten auf Werte und definiert einen modifizierten PAA $\mathcal{P}'$, nur dass man nun die Zielmenge $\mathcal{S}$ der *Zustände* (statt die der Werte) durch einen Aggregationszustand $\bullet$ und einen absorbierenden "Spülzustand" $\circ$ ersetzt. Man muss daher nicht die Operationen neu definieren, sondern die Übergangswahrscheinlichkeiten (alle Übergänge nach $\mathcal{S}$ akkumulieren sich in $\bullet$; von dort aus geht es nur nach $\circ$, wo man für immer verbleibt):

$$T'(q,q') := \begin{cases} T(q,q') & \text{wenn } q,q' \in \mathcal{Q} \setminus \mathcal{S}\,, \\ \sum_{q'' \in \mathcal{S}} T(q,q'') & \text{wenn } q \in \mathcal{Q} \setminus \mathcal{S} \text{ und } q' = \bullet\,, \\ 1 & \text{wenn } q \in \{\bullet,\circ\} \text{ und } q' = \circ\,, \\ 0 & \text{sonst}\,. \end{cases}$$

Nun ist

$$\mathbb{P}(W_{\mathcal{S}} = t) = \mathbb{P}(Q'_t = \bullet)\,.$$

Emissionen, Werte und Operationen werden hier gar nicht benötigt.

5.10 Lemma (Warten auf Zustände)**.** Sei $(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu, \theta)$ ein PAA; sei $\mathcal{S} \subset \mathcal{Q}$ eine Zielmenge von Zuständen und $W_\mathcal{S}$ die Wartezeit.

Sei $\alpha : \mathcal{Q} \to [0,1]$ eine Startverteilung auf $\mathcal{Q}$. Betrachte die Markovkette $(\mathcal{Q}, T, \alpha)$. Sei $(Q_t^\alpha)_{t\in\mathbb{N}_0}$ ihr Zustandsprozess, und sei $W_\mathcal{S}^\alpha := \min\{t \in \mathbb{N}_0 \mid Q_t' \in \mathcal{S}\}$ die Wartezeit auf einen Zustand aus $\mathcal{S}$ bei Startverteilung α.

Dann kann $\mathcal{L}(W_\mathcal{S}^\alpha)(0), \ldots, \mathcal{L}(W_\mathcal{S}^\alpha)(n)$ in $\mathcal{O}(n \cdot |\mathcal{Q}|^2)$ Zeit und mit $\mathcal{O}(|\mathcal{Q}|)$ Platz berechnet werden. Alternativ können die Werte in $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3)$ Zeit und mit $\mathcal{O}(|\mathcal{Q}|^2)$ Platz durch die Verdoppelungstechnik berechnet werden.

Mit der Wahl $\alpha := \boldsymbol{\delta}_{q_0}$ (Dirac-Verteilung in q_0) ist $W_\mathcal{S} = W_\mathcal{S}^\alpha$, und es gelten dieselben Laufzeiten.

Beweis. Laufzeit- und Platzbehauptungen folgen aus der Tatsache, dass $\mathcal{L}(Q_{t+1})$ aus $\mathcal{L}(Q_t)$ in $\mathcal{O}(|\mathcal{Q}|^2)$ Zeit berechnet werden kann, wegen

$$\mathbb{P}(Q_{t+1} = q') = \sum_{q\in\mathcal{Q}} \mathbb{P}(Q_t = q) \cdot T(q, q')\,.$$

Entsprechendes gilt für die Verdopplungstechnik. □

Mit $\alpha = \boldsymbol{\delta}_{q_0}$ erhält man die Wartezeit für das erste Auftreten eines Zustands aus $\mathcal{S}$. Man kann auch nach der Wartezeit bis zu einer Rückkehr nach $\mathcal{S}$ fragen. Ist die Markovkette aperiodisch und irreduzibel, dann hat sie eine eindeutige stationäre Zustandsverteilung, gegen die die PAA-Zustandsverteilung exponentiell schnell konvergiert. Wählen wir α als diese stationäre Verteilung, eingeschränkt auf $\mathcal{S}$, dann können wir mit Lemma 5.10 die Verteilung der Rückkehrzeiten (im Limit) berechnen.

Abschließend bemerken wir, dass sich das dritte Problem (Warten auf bestimmte Kombinationen von Zuständen und Werten) mit Hilfe einer kombinierten Konstruktion der beiden oben beschriebenen Konstruktionen lösen lässt.

5.5 Deterministische arithmetische Automaten

Bei vielen Anwendungen von PAAs sind die Emissionen gar nicht zufällig, sondern deterministisch und der Zufall kommt nur durch die zufälligen Zustandsübergänge in das Modell. Dieser Fall kommt so häufig bei der Konstruktion von PAAs vor, dass ihm hier ein eigener Abschnitt (und eine eigene Definition) gewidmet ist. Als Anwendung betrachten wir im nächsten Abschnitt die Statistik der Mustersuche in DNA-sequenzen. Hier betrachten wir allgemein die Situation, dass eine *deterministische Berechnung* auf *zufälligen Sequenzen* stattfindet.

Textmodelle für zufällige Sequenzen (insbesondere allgemeine Textmodelle mit endlichem Gedächtnis) haben wir bereits in Abschnitt 2.6 kennengelernt.

Um determinischte Berechnungen zu formalisieren, definieren wir *deterministische arithmetische Automaten*, ein deterministisches Gegenstück zu PAAs, in denen kein Zufall vorkommt.

Im Folgenden entsteht dann ein PAA durch Kombination eines Textmodells mit einem DAA.

5.11 Definition (Deterministischer arithmetischer Automat, DAA)**.** Ein *deterministischer arithmetischer Automate* (DAA) ist ein 9-Tupel

$$\mathcal{D} = \big(\mathcal{Q}^{\mathcal{D}}, q_0^{\mathcal{D}}, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q^{\mathcal{D}})_{q\in\mathcal{Q}}\big),$$

wobei $\mathcal{Q}^{\mathcal{D}}$ eine endliche Zustandsmenge ist, $q_0^{\mathcal{D}} \in \mathcal{Q}$ der Startzustand, Σ ein endliches Alphabet, $\delta : \mathcal{Q}^{\mathcal{D}} \times \Sigma \to \mathcal{Q}^{\mathcal{D}}$ eine (deterministische) Übergangsfunktion, $\mathcal{V}$ eine Wertemenge, $v_0 \in \mathcal{V}$ der Startwert, $\mathcal{E}$ eine endliche Emissionsmenge, $\eta_q \in \mathcal{E}$ die (deterministische) Emission in Zustand q, und $\theta_q^{\mathcal{D}} : \mathcal{V} \times \mathcal{E} \to \mathcal{V}$ die binäre Operation in Zustand q.

Wir definieren weiter die gemeinsame *Zustand-Wert-Übergangsfunktion*

$$\begin{aligned}&\bar{\delta} : (\mathcal{Q}^{\mathcal{D}} \times \mathcal{V}) \times \Sigma \to (\mathcal{Q}^{\mathcal{D}} \times \mathcal{V}),\\ &\bar{\delta}\big((q, v), \sigma\big) := \big(\delta(q, \sigma)\,,\, \theta^{\mathcal{D}}_{\delta(q,\sigma)}(v, \eta_{\delta(q,\sigma)})\big).\end{aligned}$$

Wir erweitern die Definitionen der Übergangsfunktionen δ und $\bar{\delta}$ induktiv von einzelnen Zeichen (Σ) auf Strings (Σ^*) in ihrem zweiten Argument, indem wir

$$\begin{aligned}\delta(q, \varepsilon) &:= q \text{ für den leeren String } \varepsilon,\\ \delta(q, x\sigma) &:= \delta(\delta(q, x), \sigma) \text{ für alle } x \in \Sigma^* \text{ und } \sigma \in \Sigma,\\ \bar{\delta}\big((q, v), \varepsilon\big) &:= (q, v),\\ \bar{\delta}\big((q, v), x\sigma\big) &:= \bar{\delta}\big(\bar{\delta}((q, v), x), \sigma\big).\end{aligned}$$

setzen.

Wir sagen, dass $\mathcal{D}$ den Wert v für die Eingabe $s \in \Sigma^*$ berechnet, wenn $\bar{\delta}\big((q_0^{\mathcal{D}}, v_0), s\big) = (q, v)$ für irgendein $q \in \mathcal{Q}^{\mathcal{D}}$ gilt, und definieren $value_{\mathcal{D}}(s) := v$.

Informell beginnt ein DAA im Zustands-Werte-Paar $(q_0^{\mathcal{D}}, v_0)$ und liest eine Sequenz von Symbolen aus Σ. Wenn im Zustand q mit Wert v das Symbol $\sigma \in \Sigma$ gelesen wird, findet im DAA ein deterministischer Zustandsübergang nach $q' := \delta(q, \sigma)$ statt, und der Wert wird zu $v' := \theta^{\mathcal{D}}_{q'}(v, \eta_{q'})$ aktualisiert, wobei die Operation und die Emission des neuen Zustands q' benutzt werden.

Da die Emission in jedem Zustand deterministisch ist, könnte man sie als Zustandsfunktion auffassen und aus der Definition des DAAs entfernen. Tatsächlich könnte man auch Werte und Operationen weglassen und einen einfachen deterministischen endlichen Automaten (DFA) auf dem Zustandsraum $\mathcal{Q}^{\mathcal{D}} \times \mathcal{V}$ definieren, der dasselbe leistet wie ein DAA. Die DAA-Definition soll aber die Verbindung zu PAAs herstellen und wird daher in der angegebenen Form verwendet.

5.6 Konstruktion eines PAAs aus einem DAA und Textmodell

Wir beschreiben nun, wie man aus einem DAA und einem allgemeinen Textmodell einen PAA konstruiert, der die Verteilung des Ergebnisses einer Sequenz von deterministischen Operationen (des DAA) auf zufälligen Texten (aus dem Textmodell) berechnet.

5.12 Lemma (DAA + Textmodell $\mapsto$ PAA)**.** Sei $(\mathcal{C}, c_0, \Sigma, \phi)$ ein allgemeines Textmodell mit endlichem Gedächtnis (Definition 2.7). Sei $\mathcal{D} = \big(\mathcal{Q}^{\mathcal{D}}, q_0^{\mathcal{D}}, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}^{\mathcal{D}}}, (\theta_q^{\mathcal{D}})_{q\in\mathcal{Q}^{\mathcal{D}}}\big)$ ein DAA. Wir definieren nun

- eine endliche Zustandsmenge $\mathcal{Q} := \mathcal{Q}^{\mathcal{D}} \times \mathcal{C}$,
- den Startzustand $q_0 := (q_0^{\mathcal{D}}, c_0)$,
- Übergangswahrscheinlichkeiten zwischen Zuständen

$$T\big((q,c),(q',c')\big) := \sum_{\sigma\in\Sigma:\, \delta(q,\sigma)=q'} \phi(c,\sigma,c'), \tag{5.9}$$

- (deterministische) Emissionswahrscheinlichkeiten für alle $(q,c) \in \mathcal{Q}$:

$$\mu_{(q,c)}(e) := \begin{cases} 1 & \text{wenn } e = \eta_q\,, \\ 0 & \text{sonst}\,, \end{cases}$$

- Operationen $\theta_{(q,c)}(v,e) := \theta_q^{\mathcal{D}}(v,e)$ für alle $(q,c) \in \mathcal{Q}$.

Es handelt sich also im Kern um eine Produktkonstruktion, bei der die Übergangswahrscheinlichkeiten aus der DAA-Übergangsfunktion und dem Textmodell entstehen. (Die Wertemenge $\mathcal{V}$, der Startwert v_0 und die Emissionsmenge $\mathcal{E}$ werden direkt vom DAA übernommen; die Emissionen und Operationen in jedem Zustand im Prinzip auch.)

Dann ist $\mathcal{P} = \big(\mathcal{Q}, q_0, T, \mathcal{V}, v_0, \mathcal{E}, \mu = (\mu_q)_{q\in\mathcal{Q}}, \theta = (\theta_q)_{q\in\mathcal{Q}}\big)$ ein PAA mit der Eigenschaft

$$\mathcal{L}(V_t) = \mathcal{L}\big(value_{\mathcal{D}}(S_0 \ldots S_{t-1})\big)$$

für alle $t \in \mathbb{N}_0$, wobei S ein Zufallstext gemäß Textmodell ist.

Beweis. Nach Definition 5.2 ist $\mathcal{P}$ ein PAA. Wie in Abschnitt 5.3 definieren wir zur Abkürzung $f_t(q,v) := \mathbb{P}(Q_t = q, V_t = v)$. Um $\mathcal{L}(V_t) = \mathcal{L}\big(value_{\mathcal{D}}(S_0 \ldots S_{t-1})\big)$ zu beweisen, zeigen wir, dass sogar

$$f_t\big((q^{\mathcal{D}},c),v\big) = \sum_{s\in\Sigma^t} \big[\![\bar{\delta}\big((q_0^{\mathcal{D}},v_0),s\big) = (q^{\mathcal{D}},v)\big]\!] \cdot \mathbb{P}(S_0 \ldots S_{t-1} = s, C_t = c) \tag{5.10}$$

für alle $q^{\mathcal{D}} \in \mathcal{Q}^{\mathcal{D}}$, $c \in \mathcal{C}$, $v \in \mathcal{V}$ und $t \in \mathbb{N}_0$ gilt. Für $t = 0$ ist (5.10) trivialerweise korrekt (dies folgt aus den Definitionen von DAAs, Textmodellen und PAAs). Für $t > 0$ wird die Behauptung mit vollständiger Induktion bewiesen. Wir dürfen also die Korrektheit für alle t'

mit $0 \leq t' < t$ voraussetzen. Nun rechnen wir

$$f_t\big(\underbrace{(q^{\mathcal{D}}, c)}_{=:q}, v\big) \tag{5.11}$$

$$= \sum_{q' \in \mathcal{Q}} \sum_{(v',e) \in \theta_q^{-1}(v)} f_{t-1}(q', v') \cdot T(q', q) \cdot \mu_q(e) \tag{5.12}$$

$$= \sum_{q' \in \mathcal{Q}} \sum_{(v',e) \in \mathcal{V} \times \mathcal{E}} [\![\theta^{\mathcal{D}}_{q^{\mathcal{D}}}(v', e) = v]\!] \cdot f_{t-1}(q', v') \cdot T(q', q) \cdot [\![\eta_{q^{\mathcal{D}}} = e]\!] \tag{5.13}$$

$$\begin{aligned} = \sum_{q'^{\mathcal{D}} \in \mathcal{Q}^{\mathcal{D}}} \sum_{c' \in \mathcal{C}} \sum_{(v',e) \in \mathcal{V} \times \mathcal{E}} & [\![\theta^{\mathcal{D}}_{q^{\mathcal{D}}}(v', e) = v]\!] \cdot [\![\eta_{q^{\mathcal{D}}} = e]\!] \\ & \cdot f_{t-1}(q', v') \cdot \sum_{\sigma \in \Sigma} [\![\delta(q'^{\mathcal{D}}, \sigma) = q^{\mathcal{D}}]\!] \cdot \varphi(c', \sigma, c) \end{aligned} \tag{5.14}$$

$$\begin{aligned} = \sum_{s \in \Sigma^{t-1}} \sum_{\sigma \in \Sigma} \sum_{q'^{\mathcal{D}} \in \mathcal{Q}^{\mathcal{D}}} \sum_{c' \in \mathcal{C}} \sum_{(v',e) \in \mathcal{V} \times \mathcal{E}} & [\![\theta^{\mathcal{D}}_{q^{\mathcal{D}}}(v', e) = v]\!] \cdot [\![\eta_{q^{\mathcal{D}}} = e]\!] \cdot [\![\delta(q'^{\mathcal{D}}, \sigma) = q^{\mathcal{D}}]\!] \\ & \cdot [\![\bar{\delta}\big((q_0^{\mathcal{D}}, v_0), s\big) = (q'^{\mathcal{D}}, v')]\!] \cdot \mathbb{P}(S_0 \dots S_{t-2} = s, C^{t-1} = c') \cdot \varphi(c', \sigma, c) \end{aligned} \tag{5.15}$$

$$\begin{aligned} = \sum_{s\sigma \in \Sigma^t} \sum_{q'^{\mathcal{D}} \in \mathcal{Q}^{\mathcal{D}}} \sum_{(v',e) \in \mathcal{V} \times \mathcal{E}} & [\![\theta^{\mathcal{D}}_{q^{\mathcal{D}}}(v', e) = v]\!] \cdot [\![\eta_{q^{\mathcal{D}}} = e]\!] \cdot [\![\bar{\delta}\big((q_0^{\mathcal{D}}, v_0), s\big) = (q'^{\mathcal{D}}, v')]\!] \\ & \cdot [\![\delta(q'^{\mathcal{D}}, \sigma) = q^{\mathcal{D}}]\!] \cdot \mathbb{P}(S_0 \dots S_{t-1} = s\sigma, C_t = c) \end{aligned} \tag{5.16}$$

$$= \sum_{s\sigma \in \Sigma^t} [\![\bar{\delta}\big((q_0^{\mathcal{D}}, v_0), s\sigma\big) = (q^{\mathcal{D}}, v)]\!] \cdot \mathbb{P}(S_0 \dots S_{t-1} = s\sigma, C_t = c). \tag{5.17}$$

In dieser Rechnung folgt Schritt (5.11)→(5.12) aus (5.6). Schritt (5.12)→(5.13) folgt aus den Definitionen von θ_q and μ_q. Schritt (5.13)→(5.14) benutzt die Definitionen von T und $\mathcal{Q}$ aus Lemma 5.12. Schritt (5.14)→(5.15) nutzt die Induktionsvoraussetzung. Schritt (5.15)→(5.16) nutzt die Gedächtnislosigkeit der Textmodelle. Der letzte Schritt (5.16)→(5.17) folgt, indem man die Iverson-Klammern, die über $q'^{\mathcal{D}}$ und (v', e) summiert werden, in eine einzelne Iverson-Klammer kombiniert. □

5.13 Bemerkung. Bei der Konstruktion in Lemma 5.12 kann man offensichtlich PAA-Zustände, die nur mit Wahrscheinlichkeit Null vom Startzustand aus erreicht werden können, weglassen.

Der so konstruierte PAA hat eine spezielle Struktur (beispielsweise sind die Emissionen deterministisch), so dass sich verbesserte Laufzeiten für die PAA-Berechnungen ergeben.

5.14 Lemma (Laufzeiten für PAA-Berechnungen aus DAA+Textmodell)**.**

1. Die Werteverteilung (oder die gemeinsame Zustands-Werteverteilung) eines PAA, der aus einem DAA und Textmodell konstruiert wurde, kann in $\mathcal{O}(n \cdot |\mathcal{Q}^{\mathcal{D}}| \cdot |\Sigma| \cdot |\mathcal{C}|^2 \cdot \vartheta_n)$ Zeit und mit $\mathcal{O}(|\mathcal{Q}^{\mathcal{D}}| \cdot |\mathcal{C}| \cdot \vartheta_n)$ Platz berechnet werden. Dieselbe Aussage gilt für die Berechnung der Verteilung der Wartezeit bis zu n Schritten.
2. Gibt es für alle Kontexte $c \in \mathcal{C}$ und Symbole $\sigma \in \Sigma$ höchstens einen möglichen Folgekontext im Textmodell, d.h., gibt es höchstens ein $c' \in \mathcal{C}$ mit $\phi(c, \sigma, c') > 0$, dann gilt eine verbesserte Laufzeit von $\mathcal{O}(n \cdot |\mathcal{Q}^{\mathcal{D}}| \cdot |\Sigma| \cdot |\mathcal{C}| \cdot \vartheta_n)$.
3. Mit der Verdopplungstechnik können die Verteilungen in $\mathcal{O}(\log n \cdot |\mathcal{Q}^{\mathcal{D}}|^3 \cdot |\mathcal{C}|^3 \cdot \vartheta_n^3)$ Zeit und mit $\mathcal{O}(|\mathcal{Q}^{\mathcal{D}}|^2 \cdot |\mathcal{C}|^2 \cdot \vartheta_n^2)$ Platz berechnet werden.

Beweis.

1. Nach Konstruktion ist $|\mathcal{Q}| \leq |\mathcal{Q}^{\mathcal{D}}| \cdot |\mathcal{C}|$. Direkte Anwendung von Lemma 5.6 liefert

   - Zeit $\mathcal{O}(n \cdot |\mathcal{Q}|^2 \cdot \vartheta_n \cdot |\mathcal{E}|) = \mathcal{O}(n \cdot |\mathcal{Q}^{\mathcal{D}}|^2 \cdot |\mathcal{C}|^2 \cdot \vartheta_n \cdot |\mathcal{E}|)$,
   - Platz $\mathcal{O}(|\mathcal{Q}| \cdot \vartheta_n) = \mathcal{O}(|\mathcal{Q}^{\mathcal{D}}| \cdot |\mathcal{C}| \cdot \vartheta_n)$.

   Während die Platzschranke die behauptete ist, ist die Zeitschranke noch verbesserungsfähig, und zwar durch eine genaue Analyse der inneren Schleife von Algorithmus 1, Zeile 4, die einen Faktor von $\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{E}|)$ beisteuert. Da der konstruierte PAA deterministische Emissionen hat, ist die Iteration über $\mathcal{E}$ unnötig. Des weiteren müssen wir nur über die Folgezustände iterieren, die in einem Schritt erreichbar sind; das sind nach Konstruktion jeweils nur $|\Sigma| \cdot |\mathcal{C}|$. Daher benötigt die innere Schleife in Zeile 4 nur $\mathcal{O}(|\Sigma| \cdot |\mathcal{C}|)$ statt $\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{E}|)$ Zeit. Dies liefert die behauptete Laufzeit.

2. Wenn es für alle Kontexte $c \in \mathcal{C}$ und Symbole $\sigma \in \Sigma$ höchstens einen möglichen Folgekontext $c' \in \mathcal{C}$ mit $\varphi(c, \sigma, c') > 0$ gibt, dann können von jedem Zustand aus höchstens $|\Sigma|$ verschiedene Zustände erreicht werden. Die behauptete Laufzeit folgt also wie unter 1. wobei ein zusätzlicher Faktor $|\mathcal{C}|$ eingespart wird.

3. Die Zeit- und Platzschranken für die Verdopplungstechnik folgen direkt aus Lemma 5.7.

□

5.7 Statistik der Mustersuche

5.7.1 Einführung

In vielen Anwendungen ist die Frage nach der statistischen Signifikanz eines häufig auftretenden Musters von Interesse: Eine häufig wiederholte DNA-Sequenz innerhalb eines Genoms, so darf man vermuten, muss eine Besonderheit (biologische Funktion, besondere Struktur) aufweisen. Natürlich kommen gewisse (kurze) Sequenzen einfach zufällig häufig vor; Dinukleotide wie `AG` findet man zum Beispiel fast überall. Die absolute Anzahl der Vorkommen ist also kein gutes Maß für die mögliche Relevanz eines Musters. Ein besseres Maß ist der p-Wert eines Musters, der sich aus der folgenden Frage ergibt. Wenn ein Muster k mal in einer Sequenz beobachtet wurde, wie groß ist die Wahrscheinlichkeit, dass mindestens k Vorkommen in einer zufälligen Sequenz auftreten? Hierbei bezieht sich "zufällig" auf ein konkretes vorgegebenes Textmodell. Die gesuchte Wahrscheinlichkeit ist der p-Wert des Musters; dieser hängt also (ggf. stark!) vom gewählten Textmodell ab. Ein GC-reiches Muster wird in einem GC-reichen Textmodell schlechtere (also höhere) p-Werte haben, weil es eben natürlicherweise häufig vorkommt. Ein niedriger p-Wert (oft kleiner als 0.05) wird als statistisch signifikant bezeichnet.

Die Untersuchung eines Motivs läuft in der Regel wie folgt ab. Gegeben sind ein Kandidatenmotiv, das untersucht werden soll, und eine (lange) Sequenz, in der das Motiv vorkommt. Wir schätzen ein Textmodell aus der Sequenz und zählen die Vorkommen das Motivs in der Sequenz. Mit Hilfe eines geeignet konstruierten PAAs berechnen wir den p-Wert (Wahrscheinlichkeit für mindestens so viele Vorkommen wie beobachtet) und entscheiden, ob das Motiv eine Bedeutung haben könnte.

Bei der DNA-Sequenzanalyse haben sich mehrere Klassen von Mustern herauskristallisiert, die häufig untersucht werden: einfache Strings, Prosite-Muster[1], Konsensus-Strings zusammen mit einem Fehlermaß und einer Fehlerumgebung, Abel'sche Muster, PWMs zusammen mit einem Schwellenwert und andere.

Alle diese Muster beschreiben letzten Endes eine endliche Menge von Strings (oft sogar Strings gleicher Länge). Daher bilden sie eine reguläre Sprache, und es gibt einen deterministischen endlichen Automaten (DFA), der diese Sprache erkennt. Die Methode, die wir in diesem Abschnitt vorstellen, geht von einem DFA aus, der zu einem DAA konvertiert wird. Daraus wird durch Kombination mit einem Textmodell ein PAA.

Das Thema der Wortstatistiken in Zufallstexten ist in den letzten Jahren sehr ausführlich unteruscht worden. Eine Übersicht enthält das Buch von Lothaire Lothaire (2005), Kapitel 6 ("Statistics on Words with Applications to Biological Sequences"), das auf einem Übersichtsartikel von Reinert et al. basiert Reinert et al. (2000). Ein mathematischer Zugang besteht darin, für die gesuchte Verteilung eine erzeugende Funktion auszurechnen, die man dann unter anderem für eine asmyptotische Analyse nutzen kann. Durch symbolische Taylorentwicklung kann man auch konkrete Werte sowie Momente der Verteilung berechnen. Ein Artikel von Régnier Régnier (2000) enthält Formeln für Erwartungswert, Varianz und höhere Momente für i.i.d. und Markov-Texte für Mengen von Mustern, die überlappend oder nichtüberlappend gezählt werden können. Nicodème et al. Nicodème et al. (2002) verfügen über einen algorithmischen Ansatz ähnlich zu PAAs, die die Verteilung der Positionen, an denen ein Vorkommen endet, berechnet; als Muster kommen vor allem reguläre Ausdrücke vor. Lladser et al. Lladser et al. (2008) haben den Terminus *Markov chain embedding* für Ansätze vorgeschlagen, die endliche Automaten mit Markovketten kombinieren. Verwandte Ansätze, um exakte p-Werte für Motive mit Hilfe von Automaten zu berechnen, stammen von Boeva et al. (2007); Nuel (2008). Ein DP-Algorithmus, um p-Werte für PWMs zu berechnen, stammt von Zhang et al. (2007).

PAAs vereinheitlichen diese Ansätze. Sie erlauben eine klare Unterscheidung zwischen verschiedenen Zählweisen von Vorkommen und erlauben allgemeine Textmodelle mit endlichem Gedächtnis.

Welche verschiedenen Zählweisen von Vorkommen gibt es? Bei der *überlappenden Zählweise* werden alle Teilstrings gezählt, die Wörtern aus der Mustermenge entsprechen. Sucht man nach `MAOAM` in `MAOAMAOAM`, ergeben sich zwei (überlappende) Vorkommen. Bei der *nichtüberlappenden Zählweise* ist die maximale Anzahl nichtüberlappender Vorkommen gefragt (in diesem Fall nur eins der beiden). Insbesondere wenn das Muster aus Wörtern verschiedener Länge besteht, wobei einige Wörter Teilstrings von anderen Wörtern sein können, ist es möglich, dass an derselben Stelle zwei oder mehr Wörter des Musters enden. Bei der überlappenden Zählweise würden diese entsprechend gezählt. Alternativ kann man nur die Positionen zählen, an denen mindestens ein Vorkommen endet (*Positionszählweise*).

5.7.2 Konstruktion von PAAs für verschiedene Motivklassen

Der Ablauf der Arbeitsschritte, um zu einem gegebenen Muster/Motiv, einem Textmodell und einer Zählweise (überlappend, nichtüberlappend, Positionen) einen PAA zu erzeugen,

[1] Diese werden in der Prosite-Datenbank benutzt (Hulo et al., 2006); eine Beschreibung der Syntax steht unter `http://www.expasy.org/tools/scanprosite/scanprosite-doc.html`.

sieht wie folgt aus.

$$\text{Muster/Motiv} \to \text{DFA} \to \text{DAA} \to \text{PAA}$$

Ein deterministischer endlicher Automat (DFA) ist bekanntlich ein 5-Tupel $(\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$. Dabei ist $\mathcal{Q}$ eine endliche Zustandsmenge, Σ ein endliches Alphabet, $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ eine Übergangsfunktion, die durch Lesen eines Zeichens des Alphabets aus einem Zustand in einen neuen Zustand übergeht, q_0 der Startzustand und $\mathcal{F} \subset \mathcal{Q}$ eine Menge von akzeptierenden Zuständen. Wie üblich erweitert man δ im zweiten Argument auf Σ^* durch induktive Fortsetzung.

Für die hier betrachteten Klassen von Mustern oder Motiven ist es einfach, einen nichtdeterministischen endlichen Automaten (NFA) zu konstruieren, der genau die durch das Muster definierte Sprache aktzeptiert. Um Vorkommen des Musters in einem (langen) Text zu finden, muss man den Automaten durch Hinzufügen eines Selbstübergangs im Startzustand, den man mit jedem Zeichen des Alphabets benutzen kann, erweitern, so dass er alle Strings akzeptiert, bei denen ein Suffix mit einem Wort des Motivs übereinstimmt (Navarro and Raffinot, 2002).

Ein NFS kann durch Teilmengenkonstruktion in einen DFA umgewandelt werden (ggf. vergrößert sich dabei die Zustandsmenge exponentiell; dies ist bei typischen Mustern jedoch die Ausnahme; häufig vergrößert sich die Zustandsmenge gar nicht. Für manche Klassen von Mustern gibt es auch optimierte Umwandlungsalgorithmen von NFA zu DFA. Bisweilen kann der DFA auch direkt konstruiert werden, siehe die Abschnitte 5.7.2 und 5.7.2.

Im nächsten Schritt wird der DFA nun in einen DAA umgewandelt. Diese Umwandlung hängt von der gewünschten Zählweise ab. Wenn ein DFA einen Text verarbeitet, befindet er sich genau dann in einem akzeptierenden Zustand, wenn (mindestens) ein Wort des Motivs gerade endet. Die Anzahl dieser Ereignisse entspricht der Anzahl der Vorkommen in der Positionszählweise. Wir definieren daher Zustandsemissionen für den DFA $(\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$ wie folgt:

$$\eta_q := \begin{cases} 1 & \text{wenn } q \in \mathcal{F}, \\ 0 & \text{sonst.} \end{cases}$$

Sind wir an der überlappenden Zählweise (statt an der Positionszählweise) interessiert, definieren wir die Emissionen etwas anders; nun muss $\eta_q \in \mathbb{N}_0$ genau die Anzahl der Treffer sein, die man zählen muss, wenn man Zustand q betritt. Dies ist dann wohldefiniert, wenn das Motiv einer endlichen Menge von Strings entspricht, wovon wir ausgehen (reguläre Ausdrücke mit Stern-Operatoren können hier Probleme verursachen).

Sind wir an der nichtüberlappenden Zählweise interessiert, definieren wir die Emissionen wie bei der Positionszählweise, müssen aber noch den Automaten modifizieren. Die von einem akzeptierenden Zustand ausgehenden Übergänge müssen so definiert werden wie die Übergänge des Startzustands q_0, d.h., wir definieren eine neue Übergangsfunktion δ' durch $\delta'(q, \sigma) := \delta(q, \sigma)$, wenn $q \notin \mathcal{F}$, und sonst $\delta'(q, \sigma) := \delta(q_0, \sigma)$. Im Folgenden bezeichnen wir die modifizierte Funktion δ' wieder mit δ.

Vor dem Schritt der Umwandlung in einen DAA ist es zweckmäßig, den DFA (wie immer man ihn konstruiert hat), zu minimieren. Mit einem Algorithmus von Hopcroft (1971) kann ein klassischer DFA in Zeit $\mathcal{O}(|\mathcal{Q}| \log |\mathcal{Q}|)$ für ein Alphabet konstanter Größe minimiert

werden. Knuutila (2001) gibt eine gute Einführung in die Methodik und beschreibt eine Variante, die in Zeit $\mathcal{O}(|\Sigma| \cdot |\mathcal{Q}| \log |\mathcal{Q}|)$ läuft, wenn das Alphabet keine konstante Größe hat. Hopcroft's Algorithmus kann leicht modifiziert werden, um für DFAs mit Emissionen (die von 0 oder 1 verschieden sein können im Fall der überlappenden Zählweise) zu minimieren. Die initiale Partition der Zustände, die im klassischen Fall zwischen akzeptierenden und nichtakzeptierenden Zuständen unterscheidet, wird nun ersetzt durch eine mehrelementige Partition, die die Zustände anhand ihres Emissionswerts trennt.

Um den DAA $\mathcal{D} = \big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big)$ vollständig zu spezifizieren, müssen wir noch die Wertemenge $\mathcal{V}$, den Startwert v_0 und die Operationen festlegen. Bei der Anwendung des Zählens ist das aber klar: $\mathcal{V}$ ist $\mathbb{N}$ (oder eine endliche Menge $\{0, \dots, M\}$, wobei die letzte Zahl M dann die Bedeutung $\geq M$ erhält, $v_0 := 0$ und die Operation ist einfach die Addition (oder die bei M abgeschnittene Addition).

Im Falle der Positionszählweise oder der nichtüberlappenden Zählweise, aber auch bei der uberlappenden Zählweise, wenn alle im Motiv enthaltenen Worter die gleiche Länge haben, ist stets $\vartheta_n = \Theta(n)$.

5.15 Satz. *Sei ein DFA $(\mathcal{Q}, \Sigma, \delta, q_0)$ mit zusätzlichen zählenden Emissionen $(\eta_q)_{q\in\mathcal{Q}}$ gegeben, sowie ein Textmodell $(\mathcal{C}, c_0, \Sigma, \varphi)$. Sei $(S_t)_{t\in\mathbb{N}_0}$ ein Zufallstext gemäß diesem Modell. Dann kann die (bei M abgeschnittene) Verteilung der akkumulierten Zähler*

$$\mathcal{L}\left(\min\left\{M\,,\,\sum_{i=0}^{n-1}\eta_{\delta(q_0,S_0\dots S_i)}\right\}\right) \tag{5.18}$$

in $\mathcal{O}(n \cdot |\mathcal{Q}| \cdot |\mathcal{C}|^2 \cdot |\Sigma| \cdot M)$ Zeit und mit $\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{C}| \cdot M)$ Platz berechnet werden.

Wenn für alle Textkontexte $c \in \mathcal{C}$ und Buchstaben $\sigma \in \Sigma$ höchstens ein Folgekontext möglich ist (d.h., es höchstens ein $c' \in \mathcal{C}$ mit $\varphi(c, \sigma, c') > 0$ gibt), dann ist die Laufzeit sogar beschränkt durch $\mathcal{O}(n \cdot |\mathcal{Q}| \cdot |\mathcal{C}| \cdot |\Sigma| \cdot M)$.

Alternativ kann (5.18) *in $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot |\mathcal{C}|^3 \cdot M^2)$ Zeit und mit $\mathcal{O}(|\mathcal{Q}|^2 \cdot |\mathcal{C}|^2 \cdot M)$ Platz berechnet werden.*

Beweis. Aus dem DFA $(\mathcal{Q}, \Sigma, \delta, q_0)$ und den Zählemissionen $(\eta_q)_{q\in\mathcal{Q}})$ konstruieren wir den DAA $\mathcal{D} = \big(\mathcal{Q}, q_0, \Sigma, \delta, \mathcal{V}, v_0, \mathcal{E}, (\eta_q)_{q\in\mathcal{Q}}, (\theta_q)_{q\in\mathcal{Q}}\big)$ mit Wertemenge $\mathcal{V} := \{0, \dots, M\}$, Startwert $v_0 := 0$ und passender Emissionsmenge $\mathcal{E} := \{\eta_q : q \in \mathcal{Q}\}$. Die Operation in jedem Zustand ist die bei M abgeschnittene Addition:

$$\theta_q(v, e) := \begin{cases} M & \text{wenn } v + e \geq M\,, \\ v + e & \text{sonst} \end{cases}$$

für alle $q \in \mathcal{Q}$. Für diesen DAA $\mathcal{D}$ gilt

$$\mathit{value}_{\mathcal{D}}(s) = \min\left\{M\,,\,\sum_{i=0}^{n-1}\eta_{\delta(q_0,s[\dots i])}\right\}$$

für alle Strings $s \in \Sigma^*$.

Wir wenden nun Lemma 5.12 auf diesen DAA und das Textmodell an und erhalten den PAA. Die Laufzeit- und Platzschranken für den grundlegenden Algorithmus folgen direkt.

Um die genannten alternativen Schranken für den Verdopplungsalgorithmus zu erhalten, nämlich $\mathcal{O}(\log n \cdot |\mathcal{Q}|^3 \cdot |\mathcal{C}|^3 \cdot M^2)$ Zeit und $\mathcal{O}(|\mathcal{Q}|^2 \cdot |\mathcal{C}|^2 \cdot M)$ Platz, nutzen wir aus, dass die Operationen (im wesentlichen) Additionen sind. Daher ist $U^{(t)}(q_1, q_2, v_1, v_2) = U^{(t)}(q_1, q_2, v_3, v_4)$, wenn $v_1 \leq v_2 < M$, $v_3 \leq v_4 < M$ und $v_2 - v_1 = v_4 - v_3$. Daher genügt es, alle Berechnungen auf den Fall $v_1 = 0$ einzuschränken; so spart man einen Faktor von $|\mathcal{V}| = \mathcal{O}(M)$ gegenüber dem Standardfall. Die Spezialfälle des Abschneidens ($v_2 = M$ oder $v_4 = M$) können in derselben Zeit berücksichtigt werden. □

Wir diskutieren nun einige wichtige Klassen von Motiven: endliche Mengen von (normalen) Strings, endliche Mengen von verallgemeinerten Strings und Prosite-Muster.

Endliche Mengen von Strings

Ist das Motiv explizit als endliche Menge von Strings gegeben, konstruiert man am besten den zugehörigen Aho-Corasick-Automaten. Dies ist genau der die Vorkommen zählende DFA. Die Konstruktion ist in Linearzeit mit dem Original-Algorithmus von Aho and Corasick (1975) möglich oder mit einem neueren eleganten Algorithmus, der einen Suffixbaum der reversen String benutzt (Dori and Landau, 2006). Die Emissionen η_q werden durch die Ausgabe-Funktion des Aho-Corasick-Automaten gegeben.

Endliche Mengen verallgeminerter Strings

Verallgemeinerte Strings sind endliche Sequenzen von *Mengen* von Symbolen über einem Alphabet Σ, zum Beispiel `[abc][ac][ab]` (passt zum Beispiel auf die Texte `aaa` und `ccb`, aber nicht auf `aba`). Man kann natürlich zu jedem verallgemeinerten String alle (normalen) Strings aufzahlen; jedoch kann man einfacher direkt einen NFA konstruieren, der alle Texte erkennt, die mit einer Instanz eines verallgemeinterten Strings enden: Der NFA zu einem verallgemeinerten String ist einfach eine lineare Kette von Zuständen, bestehend aus einem Startzustand plus ein Zustand für jede Position. Die Positionen sind mit den entsprechenden Buchstabenmengen verbunden. Der Startzustand erhält zusätzlich einen Σ-Selbstübergang.

Der NFA für eine *Menge* von verallgemeinerten Strings wird konstruiert, indem man die individuellen Startzustände der einzelnen verallgemeinerten Strings zu einem gemeinsamen Startzustand verbindet.

Im nächsten Schritt muss aus dem NFA ein DFA erstellt werden; dies gelingt mit der klassischen Teilmengenkonstruktion und ggf. anschließender Minimierung. Marschall (2011) hat für den Fall einer Menge von verallgemeinerten Strings eine effiziente direkte Konstruktion beschrieben, die stets zu einem minimalen DFA führt.

Die Zähl-Emissionen $\eta_s tate$ erhält man wie folgt: Durch die Teilmengenkonstruktion entspricht jeder Zustand q des DFA einer Teilmenge B_q von NFA-Zuständen. Die Anzahl $|B_q$ entspricht der Anzahl der passenden verallgemeinerten Strings, die enden, wenn der DFA-Zustand q erreicht wird, so dass $\eta_s tate = |B_q|$.

Prosite-Muster

Prosite ist eine Datenbank von biologisch bedeutungsvollen Aminosäuremustern (Hulo et al., 2006). Prosite-Muster sind verallgemeinerte Strings mit der Erweiterung, dass zu jederositi-on ein Bereich von Vielfachheiten angegeben werden kann. Zur besseren Trennung werden einzelne Elemente eines Musters durch Bindestriche getrennt. Zum Beispiel beschreibt das Prosite-Muster `A-x(2,3)-C` ein `A`, gefolgt von zwei oder drei beliebigen Zeichen, gefolgt von `C`. Wir übersetzen ein Prosite-Muster in eine Menge von verallgemeinerten Strings, im Beispiel `AxxC` and `AxxxC`, wobei das `x` eine beliebige Aminosäure bedeutet, und verfahren dann wie oben.

5.7.3 Wartezeiten für Muster

Die Wartezeit bis zum ersten Vorkommen eines Musters entspricht der Wartezeit, bis ein Zustand erreicht wird, der eine Zahl größer Null emittiert. Daher kann die Verteilung der Wartezeit mit Lemma 5.10 berechnet werden.

5.8 Statistik und Optimierung von Pyrosequencing

Den Prozess der Bestimmung der Sequenz (String aus `A,C,G,T`) eines DNA-Moleküls nennt man *DNA-Sequenzierung*. Eine Technologie (Roche/454), die man auch als Pyrosequencing bezeichnet, lässt die vier verschiedenen fluoreszierend markierten Nukleotide nacheinander in mehreren Zyklen über entprechend vorbereitete einsträngige DNA-Proben (*reads*) zu fließen und misst mit einem Lichtsignal, welche Nukleotide an welche DNA-Probe binden. Eine andere Technologie (IonTorrent) funktioniert ähnlich, misst aber die Änderung des PH-Werts. Die Signalintensität ist in der Regel proportional zur Anzahl der gleichartigen eingebauten Nukleotide, auch als *Homopolymer-Länge* bezeichnet. Ein Homopolymer der Länge 1 (z.B., ein `A`) führt nach Konvention zu einer durchschnittlichen Signalintensität von 100.

Ein Roche/454 Instrument benutzt 200 Zyklen mit der Nukleotidreihenfolgte `TACG`; es gibt also insgesamt 800 Nukleotidflüsse, für die jeweils die Signalintensität gemessen wird.

Ein Beispiel: Typische Messungen bei einem read `TCCCGGAG...` würden wie folgt beginnen (Reihenfolge `TACG`): $(100, 0, 300, 200, 0, 100, 0, 100, \dots)$, wobei die gemessenen Werte mehr oder weniger stark verrauscht sein werden.

Eine interessante Frage ist nun, wie viele Nukleotide man mit 200 Zyklen (bzw. 800 Flüssen; ein Zyklus muss nicht notwendigerweise eine Länge haben, die ein Vielfaches von 4 ist) sequenzieren kann. Ein Fluss kann null, ein oder mehrere Nukleotide sequenzieren. Die Verteilung der Länge eines sequenzierten Reads hängt damit vor allem vom Genom und der Flussreihenfolge ab. Die statistischen Eigenschaften eines Genoms können durch ein Textmodell beschrieben werden. Wir verwenden PAAs, um die Frage nach der Längenverteilung zu beantworten.

Wir konstruieren zunächst einen DAA, der eine andere Frage beantwortet: Gegeben eine Flussreihenfolge $d = d[0] \dots d[\ell - 1]$, die beliebig oft wiederholt werden kann, wie viele Flüsse werden benötigt, um eine gegebene Nukleotidsequenz zu sequenzieren? In jedem Schritt verarbeitet der DAA ein Nukleotid der Sequenz, und der emittierte Wert gibt an, wie viele Flüsse

es (seit dem letzten sequenzierten Nukleotid) dauert, bis es sequenziert wurde. Gehört das Nukleotid beispielsweise zum gleichen Homopolymer, ist also gleich dem vorigen Nukleotid, ist diese Zeit gleich Null. Die Operation ist die (abgeschntten) Addition.

Der resultierende PAA (nach Kombination des DAA mit dem Genom-Textmodell) berechnet dann die Verteilung der Flüsse, die benötigt werden, um Sequenzen beliebiger vorgegebener Längen zu Sequenzieren. Das entsprechende Wartezeitproblem hingegen berechnet das, was uns interessiert: die Verteilung der Anzahl der Nukleotide, bis ein Wert von f Flüssen erreicht wurde. Ist also der PAA konstruiert, so erhält man die gesuchte Verteilung mit Lemma 5.9 für jede Wahl von f.

Wir definieren den DAA mit der Zustandsmenge $\mathcal{Q}^{\mathcal{D}} := (\{0, \ldots, \ell - 1\} \times \{0, \ldots, \ell - 1\}) \cup \bigcup_{i=-1}^{\ell-1} \{(-1, i)\}$ und dem Startzustand $q_0^{\mathcal{D}} := (-1, -1)$ mit der folgenden Semantik: Zustand (i, j) mit bedeutet, dass die letzten beiden sequenzierten Nukleotide in den Flüssen mit den Indizes i und j (in der Flussreihenfolge d der Länge ℓ, die zyklisch durchlaufen wird) angehängt wurden, also $d[i]$ und $d[j]$ waren. Der Zustand (i, j) emittiert die Anzahl der Flüsse, die benötigt werden, um von Flussindex i zu Flussindex j zu gelangen, also $\eta_{(i,j)} = (j - i) \mod \ell$. Die Wertemenge ist $\mathcal{V} := \{0, 1, \ldots, f + 1)\}$ mit $v_0 := 0$. Die Operation in jedem Zustand ist die Addition, die bei $f + 1$ abschneidet (typischerweise $f = 800$ bei der 454-Sequenzierung). Obwohl die Zustandsmenge zunächst die Größe $\mathcal{O}(\ell^2)$ hat, existieren höchstens $\mathcal{O}(|\Sigma|\ell)$ erreichbare Zustände, da das Übergangsziel jedes Zustands (i, j) nur von j und dem nächsten Zeichen abhängt: Es ist $\delta((i, j), \sigma)) = (j, k)$, wobei k der kleinste Index $k > j$ mit $d[k] = \sigma$ ist, wobei man wieder bei 0 anfängt, wenn d zu Ende ist.

Lemma 5.12 liefert uns nun einen PAA für ein beliebiges Textmodell. Wir interessieren uns für die Längenverteilung des sequenzierten DNA-Reads (bis zu einem gewünschten Maximum n). Dazu definieren wir die Zielmenge $\mathcal{T} := \{f + 1\}$ (für Flüsse). Die gesuchte Readlänge ist dann eins weniger als die Wartezeit

$$W_{\mathcal{T}} = \min \{t \in \mathbb{N}_0 \mid V_t = f + 1\}.$$

Die Analyse erfolgt durch Anwenden von Lemma 5.14 mit $\Sigma = \mathcal{O}(1)$, $|\mathcal{C}| = \mathcal{O}(1)$, $|\mathcal{Q}^{\mathcal{D}}| = \mathcal{O}(\ell)$ und $\vartheta_n = \mathcal{O}(\ell n)$ resultiert in einer Laufzeit von $\mathcal{O}(n^2\, \ell^2)$ und Platzbedarf von $\mathcal{O}(n\ell^2)$.

Als eine Anwendung können wir beispielsweise ein Markovmodell zweiter Ordnung aus dem GC-reichen Genom des Bakteriums *S. meliloti* schätzen und die erwartete Readlänge für alle Flussreihenfolgen der Länge 4 (Permutationen des Alphabets) berechnen. Die Standardreihenfolge `TACG` liefert eine Verteilung mit Erwartungswert 513.9 bp, während die Verteilung mit der Reihenfolge `TCGA` eine Verteilung mit Erwartungswert 559.3 bp liefert, eine Verbesserung von etwa 8.9 %.

KAPITEL 6

Phylogenetik: Merkmalsbasierte Methoden

6.1 Einführung und Grundlagen

Fragestellung. Phylogenetische Bäume (auch evolutionäre Bäume) stellen die evolutionären Beziehungen zwischen einer Menge von Objekten (häufig Spezies oder Gruppen von Spezies) dar. Die Blätter des Baum repräsentieren die zur Zeit existierenden (und damit beobachtbaren) Spezies (oder Gruppen von Spezies). Man spricht in diesem Zusammenhang auch von *operational taxonomic unit* (OTU) oder einfach Taxon (Plural: Taxa).

Die inneren Knoten sind stets verzweigend (haben einen Ausgrad von mindestens 2) und repräsentieren die letzten gemeinsamen Ahnen vor einem Speziations-Ereignis (Auftrennung einer Spezies in zwei). Da diese Ahnen im Normalfall nicht mehr existieren, sind Daten für sie (im Gegensatz zu den Blättern) in der Regel nicht verfügbar.

Das Ziel einer phylogenetischen Untersuchung ist, aufgrund der Merkmalsausprägungen der heutigen Spezies auf die Baumtopologie und auf die Merkmalsausprägungen bei den Ahnen zu schließen. Aufgrund der fehlenden Daten aus der Vergangenheit lassen sich nur Mutmaßungen anstellen, deren Plausibilität durch bestimmte Annahmen vewertet werden kann.

Da die Speziation (Auftrennung von Arten) ein anschaulich schwer vorstellbarer Prozess ist, machen wir noch einige Bemerkungen dazu. Zunächst lässt sich formal nur schwer definieren, was eigentlich eine Spezies (Art) genau ist. Häufig definiert man eine Art als die Menge aller Individuen, die sich miteinander paaren und fruchtbaren Nachwuchs zeugen können; das führt aber im Detail auf Probleme. In jedem Fall bezieht sich der Begriff Art immer auf eine Population von Individuen, nicht auf ein einzelnes Individuum. Veränderungen (Mutationen) des Genoms finden aber auf individueller Ebene statt, so dass eine Speziation immer damit beginnen muss, dass es von der "Norm" abweichende Individuen gibt. Zur Entwicklung einer neuen Art müssen diese in der Regel räumlich von ihren bisherigen Artgenossen getrennt

werden, so dass sich die beiden Populationen getrennt entwickeln können. Im Laufe der Zeit häufen sich die genetischen Unterschiede zwischen den Teilpopulationen, so dass eine Paarung auch dann nicht mehr funktionieren würde, wenn sie ihren Lebensraum wieder miteinander teilen würden.

Bäume. Wir setzen voraus, dass folgende grundlegende Begriffe zu Bäumen bekannt sind: (ungerichteter / gerichteter) Graph, Knoten, Kanten, Grad / Eingrad / Ausgrad eines Knoten, Zusammenhang, Pfad, Kreis oder Zyklus, einfacher Kreis (benutzt jede Kante höchtens einmal), Kantengewichte, Länge eines Pfades, Kanten- und Knotenlabel, Baum, Charakterisierung eines ungerichteten Baums: azyklisch und zusammenhängend, Wurzel eines Baums, innerer Knoten eines Baums, Blatt eines Baums.

Die hier betrachteten Bäume haben verschiedene Beschriftungen (Label) in den Blättern, d.h. die Blätter sind unterscheidbar.

Wir unterscheiden hier zwischen ungerichteten *ungewurzelten Bäumen* und gerichteten *gewurzelten Bäumen*, in denen per Konvention alle Kanten von der Wurzel zu den Blättern zeigen. Vergisst man in einem gewurzelten binären Baum die Wurzel (und klebt die beiden offenen Kanten zusammen), erhält man einen ungewurzelten Baum, in dem die ursprünglichen Kantenrichtungen unbekannt sind.

Ein gewurzelter Baum ist ein *Binärbaum*, wenn die Wurzel und jeder innere Knoten Ausgrad 2 hat. Ein ungewurzelter Baum ist ein Binärbaum, wenn jeder innere Knoten den Grad 3 hat. Ein phylogenetischer Baum heißt auch *vollständig aufgelöst*, wenn er ein Binärbaum ist. Ein Knoten, der die Binärbaum-Bedingungen erfüllt, heißt *Bifurkation*. Ein Knoten, der die Bedingungen verletzt, heißt *Multifurkation* oder *Polytomie*. Multifurkationen entstehen, wenn man die genaue Reihenfolge von Speziations-Ereignissen nicht genau festlegen kann.

Unter der Topologie eines Baumes verstehen wir sein Verzweigungsmuster; wir präzisieren dies gleich. Wir bemerken dazu, dass die (ausgehenden) Kanten in einem Knoten *keine Reihenfolge* haben (innere Knoten und die Wurzel verhalten sich also wie p-Knoten in einem pq-Baum). Äquivalent zum Verzweigungsmuster ist bei gewurzelten Bäumen die Mengen der Blätter in den Unterbäumen, die in den inneren Knoten wurzeln. Bei ungewurzelten oder gewurzelten Bäumen entspricht das der Menge der *Splits*. Ein Split ist eine Partition der Blätter in zwei Mengen, die entsteht, wenn man eine Kante im Baum löscht, so dass der Baum in zwei Zusammenhangskomponenten zerfällt. Zwei Bäume mit beschrifteten Blättern sind genau dann gleich (d.h., haben das gleiche Verzweigungsmuster), wenn sie die gleiche Split-Menge besitzen.

Ein gewurzelter Baum lässt sich kompakt im sogenannten PHYLIP- oder NEWICK-Format angeben; siehe auch `http://evolution.genetics.washington.edu/phylip.newicktree.html`. Die Grundidee dabei ist, dass der String (x, y) einen inneren Knoten mit Unterbäumen x und y darstellt. Dabei können x und y entweder wieder innere Knoten oder Blattbezeichnungen sein. Blätter und Spezies bezeichnen wir hier der Kürze wegen mit A, B, C, etc.

Es sollte nun klar sein, dass (A,(B,C)), (A,(C,B)), ((B,C),A), ((C,B),A) jeweils denselben gewurzelten Baum darstellen, der dadurch charakterisiert ist, dass die Blätter B und C einen gemeinsamen Unterbaum bilden, also zueinander näher verwandt sind als zu A.

Bei ungewurzelten Bäumen legt man die Wurzel an irgend eine Kante und gibt die Klammerdarstellung des entsprechend gewurzelten Baums an. Natürlich bezeichnen dann sehr verschiedene Darstellungen den gleichen ungewurzelten Baum. Zum Beispiel ist dann (((A,B),C),(D,E)) = (A,(B,(C,(D,E)))).

Beim Zeichnen von gewurzelten Bäumen wird die Wurzel normalerweise oben gezeichnet, die Blätter unten. Die y-Achse repräsentiert dann die Zeit: Oben ist die Vergangenheit, unten die Gegenwart. Die x-Achse (also die Reihenfolge der Blätter) hat keine Bedeutung.

Abzählen ungewurzelter Binärbäume. Wir stellen nun die Frage, wie viele verschiedene ungewurzelte Binärbäume U_n für n Taxa existieren. Für $n = 3$ ist die Frage leicht zu beantworten: Offensichtlich ist $U_3 = 1$; es gibt genau einen inneren Knoten, von dem die 3 Taxa sternförmig abzweigen, und damit 3 Kanten. Um zu ermitteln, wie groß U_{n+1} für $n \geq 3$ ist, argumentieren wir wie folgt: In jeden der U_n Bäume können wir das $(n+1)$-te Taxon in jeder der $2n-3$ Kanten einfügen (durch Einfügen eines neuen inneren Knoten, der die existierende Kante teilt, und durch weiteres Einfügen einer neuen Blattkante). Jede dieser $U_n \cdot (2n-3)$ Möglichkeiten führt auf eine unterschiedliche Baumtopologie. Weitere Topologien gibt es nicht. Daher gilt

$$U_{n+1} = U_n \cdot (2n-3).$$

Wenn wir nun nach der Anzahl W_n von gewurzelten Binärbaumtopologien fragen, dann stellen wir fest, dass die Wahl der Position der Wurzel genau der Wahl einer weiteren Kante entspricht, als würde man ein $(n+1)$-tes Blatt einfügen wollen. Daher ist $W_n = U_{n+1}$.

Die folgende Tabelle zeigt U_n und W_n für einige kleine Werte von n.

n	3	4	5	6	7	8
U_n	1	3	15	105	945	10395
W_n	3	15	105	945	10395	135135

Insgesamt ergibt sich folgender Satz, der sich nun leicht per Induktion beweisen lässt.

6.1 Satz. *Sei $n \geq 3$. Es gibt genau $U_n = \prod_{i=3}^{n} (2i-5)$ ungewurzelte Binärbäumtopologien mit n verschiedenen Blättern. Es gibt genau $W_n = U_{n+1}$ gewurzelte Binärbaumtopologien mit n verschiedenen Blättern.*

Es folgt, dass die Anzahl der Binärbaumtopologien überexponentiell mit n wächst. Erlaubt man zusätzlich Multifurkationen, ist die Anzahl der Baumtopologien noch bedeutend höher.

Will man erschöpfend alle (ungewurzelten oder gewurzelten) Baumtopologien mit n Blättern aufzählen, kann man dies einfach mit einer rekursiven Funktion tun, die die obige Argumentation nachvollzieht und nacheinander die Blätter (und ggf. zum Schluss die Wurzel) an allen möglichen Positionen einfügt.

Merkmale. Häufig stehen wir vor dem oben genannten Problem, dass wir zu einer Menge von Taxa bestimmte Merkmale beobachten können und daraus rekonstruieren wollen, wie diese Taxa evolutionär zusammenhängen. Wir formalisieren zunächst den Merkmalsbegriff.

6.2 Definition (Merkmal; Ausprägung oder Zustand)**.** Ein *Merkmal* ist ein Paar (λ, C) aus einem Namen λ und einer Menge C. Jedes $c \in C$ heißt *Ausprägung* oder *Zustand* des Merkmals.

Es gibt verschiedene Arten von Merkmalen;

binär Die Zustandsmenge ist äquivalent zu $C = \{0, 1\}$, wobei konventionell die 1 die Existenz einer Eigenschaft anzeigt. Ein Beispiel ist $\lambda =$ "Existenz eines zentralen Nervensystems".

linear geordnet Die Zustandsmenge ist linear geordnet, z.B. $\lambda =$ "Anzahl der Extremitäten (Arme, Beine, etc.)".

kategoriell Die Menge C ist endlich, und es gibt keine offensichtliche Ordnung der Zustände. Ein Beispiel ist ein DNA-Nukleotid an einer bestimmten Position in einem Gen; hier ist $C = \{$ A,C,G,T $\}$.

Im Folgenden gehen wir davon aus, dass zu n Taxa die Ausprägungen von m Merkmalen gegeben sind (z.B. in Form einer $n \times m$-Tabelle oder -Matrix). In diesem Zusammenhang bezeichnen wir auch (nicht ganz korrekt) eine Tabellenspalte als *Merkmal* (eigentlich: Vektor der Ausprägungen eines Merkmals in allen Taxa) und eine Tabellenzeile als *Taxon* (eigentlich: Vektor der Ausprägungen aller Merkmale im Taxon).

6.3 Beispiel (Fahrzeuge)**.** Fahrräder, Motorräder, Autos und Dreiräder sind Fahrzeuge. Wir betrachten die Merkmale "Anzahl der Räder" mit Zustandsmenge $\mathbb{N}_0$ (linear geordnet) und "Existenz eines Motors" (binär). Eine entsprechende Merkmalsausprägungsmatrix (Taxa $\times$ Merkmale) kann jede/r leicht aufstellen. ♡

6.2 Perfekte Phylogenien

Ein wesentliches Argumentationsprinzip in der Phylogenetik ist, dass sich Merkmale nicht beliebig ändern können. Der Zustand aller Merkmale ist durch die Erbinformation codiert, sie sich sehr stabil unverändert weitervererbt. Dabei kommt es nur gelegentlich zu sinnvollen Mutationen, die zu anderen Merkmalsausprägungen führen. Daher kann man (approximativ) davon ausgehen, dass jedes Merkmal im Lauf der Evolution nur einmal "erfunden" wurde, sich also zufällig herausgebildet hat.

Dies führt auf die Begriffe der Kompatibilität eines Merkmals mit einer Baumtopologie und einer Perfekten Phylogenie (PP).

6.4 Definition (Kompatibilität)**.** Ein Merkmal mit Ausprägungsmenge C heißt *kompatibel* mit einer Baumtopologie $T = (V, E)$, wenn man alle inneren Knoten (und ggf. die Wurzel) so mit Ausprägungen beschriften kann, dass für alle $c \in C$ gilt: Die Knotenmenge $V_c := \{ v \in V \mid label(v) = c \}$ induziert eine einzige Zusammenhangskomponente.

6.5 Definition (Perfekte Phylogenie)**.** Eine Baumtopologie $T = (V, E)$ heißt *Perfekte Phylogenie* für eine Ausprägungsmatrix, wenn sie kompatibel mit jedem Merkmal darin ist.

Natürlich stellt sich die Frage, ob man einer Ausprägungsmatrix ansehen kann, ob eine perfekte Phylogenie existiert.

6.6 Problem (Problem der Perfekten Phylogenie)**.** Gegeben eine $n \times m$-Merkmalsausprägungsmatrix, gibt es eine perfekte Phylogenie dazu? •

Im Allgemeinen ist das Problem NP-schwer (**TODO: Zitat**). Unter der Voraussetzung, dass alle Merkmale binär sind und wir einen gewurzelten Baum suchen, der in der Wurzel ausschließlich die Null-Ausprägungen aufweist (bisher kein Merkmal erfunden), dann gibt es eine einfache Charakterisierung solcher Matrizen (**TODO: Zitat Gusfield**).

6.7 Satz (Charakterisierung Perfekter Phylogenien)**.** *Sei $M = (M_{ij}) \in \{0,1\}^{n \times m}$ eine binäre Merkmalsausprägungsmatrix. Zu Spalte j sei $O_j := \{ i : M_{ij} = 1 \}$ die Menge der Taxa (oder Objekte), bei denen Merkmal j ausgeprägt ist. Dann gibt es zu M genau dann eine PP, wenn für je zwei Spalten j, k gilt: $O_j \subseteq O_k$ oder $O_k \subseteq O_j$ oder $O_k \cap O_j = \{\}$.*

Den Beweis sparen wir uns und geben nur kurz die Idee an: Dass die genannte Bedingung in einer PP erfüllt ist, ist klar. Umgekehrt: Gilt $O_j \subset O_k$, dann wird Merkmal j "nach" Merkmal k erfunden, und man kann die Taxa so anordnen, dass alle mit Merkmal k einen Teilbaum bilden und die mit Merkmal j einen Unterbaum darin. Gilt $O_j \cap O_k$, treten die Merkmale j, k nie zusammen auf, d.h. man kann die Taxa so anordnen, dass diejenigen mit Merkmal j und diejenigen mit Merkmal k jeweils einen Unterbaum bilden und diese beiden Unterbäume disjunkt sind.

Ein naiver Test der Bedingung kostet Zeit $\mathcal{O}(nm^2)$, da man alle Paare von Spalten testen muss. Durch Sortieren der Spalten mit Radix-Sort und weitere Tricks ist aber ein Test (und die Konstruktion einer PP, wenn sie existiert) sogar in $\mathcal{O}(mn)$ Zeit möglich. Prinzipiell erstellt man, um eine PP zu rekonstruieren, das Hasse-Diagramm der Inklusionsbeziehungen der O_j-Mengen: Jedes O_j bildet einen Knoten, Kanten zieht man vom direkten Vorgänger zum direkten Nachfolger bezüglich der Teilordnung, die durch $\subseteq$ gegeben ist. Aufgrund der Bedingung im Satz ist das Hasse-Diagramm ein Baum, aus dem man die PP leicht erhalten kann.

Wir betrachten noch kurz den Fall, dass die binäre Matrix M in einer Form gegeben ist, zu der zwar prinzipiell eine PP existiert, aber die Bedingung verletzt ist, dass alle Merkmale in der Wurzel die Ausprägung 0 haben. In diesem Fall nützt Satz 6.7 zunächst nicht. Eine (naive) Lösung wäre zu versuchen, gewisse Merkmale zu invertieren. Möchte man aber alle Kombinationen von Merkmalen durchprobieren, müsste man das PP 2^m-mal lösen. Glücklicherweise gibt es ein einfaches Kriterium, das sagt, welche Merkmale man invertieren muss (ohne Beweis).

6.8 Lemma. Gibt es zu einer binären Matrix M überhaupt eine PP, und invertiert man die Spalten, in denen mehrheitlich der Zustand 1 vorkommt, dann erfüllt die resultierende Matrix M' die Bedingungen zu Satz 6.7.

6.3 Maximum Parsimony

6.3.1 Motivation

Wenn man m echte Merkmale in n Spezies beobachtet und daraus eine Perfekte Phylogenie zu rekonstruieren versucht, stellt man häufig fest, dass es eine solche nicht gibt. Es gibt mehrere Möglichkeiten, die Forderung nach "Perfektion" aufzuweichen:

1. Die gegebenen Daten werden als fehlerhaft betrachtet; und wir suchen nach einer Datenmatrix, die der gegebenen möglichst ähnlich ist, zu der es aber eine Perfekte Phylogenie gibt. Hierzu müssen wir ein Abstandsmaß zwischen zwei Merkmalsausprägungsmatrizen definieren. Wir kommen in Abschnitt **??** darauf zurück.

2. Dass im Laufe der Evolution keine Merkmalsausprägung mehrfach unabhängig erfunden wird (wie es ja eine Perfekte Phylogenie voraussetzt), ist erwiesenermaßen nicht immer wahr. (**TODO: Beispiel?**) Da es aber unwahrscheinlich ist, dass die Evolution zweimal zufällig das Gleiche hervorbringt, sollten solche Ereignisse sehr selten sein, gewissermaßen *sparsam* verwendet werden. Das englische Wort für Sparsamkeit ist *parsimony*. Bei Maximum Parsimony Problemen versuchen wir also eine Baumtopologie und eine Belegung der inneren Knoten zu finden, so dass möglichst wenig Änderungen von Merkmalsausprägungen entlang von Kanten stattfinden.

Man beachte den grundlegenden Unterschied, durch den sich die beiden Ansätze unterscheiden. Im ersten Fall werden nur Perfekte Phylogenien für wahr gehalten; und wenn die Daten dem widersprechen, wird das durch Fehler in den Daten erklärt. Im zweiten Fall (Maximum Parsimony) wird das Perfekte Phylogeniemodell in Frage gestellt.

6.3.2 Problemstellung

Man kann verschiedenen Zustandswechseln eines Merkmals verschiedene Kosten zuweisen und erhält dann folgende formale Probleme.

6.9 Definition (Kostenfunktion)**.** Eine *Kostenfunktion* beim Kleinen Parsimony-Problem für ein Merkmal mit möglichen Ausprägungen aus einer Menge X ist eine Funktion $\text{cost} : X \times X \to \mathbb{R}_{\geq 0}$, so dass $\text{cost}(x, x') = 0$ genau dann wenn $x = x'$.

Gelegentlich setzen wir sogar stärker voraus, dass die Kostenfunktion eine Metrik ist, d.h. dass darüberhinaus Symmetrie und die Dreiecksungleichung gelten; wir sprechen dann von einer *Kostenmetrik*:

- Symmetrie: $\text{cost}(x, y) = \text{cost}(y, x)$ für alle $x, y \in X$
- Dreiecksungleichung: $\text{cost}(x, y) \leq \text{cost}(x, z) + \text{cost}(z, y)$ für alle $x, y, z \in X$

6.10 Problem (Kleines Parsimony-Problem für ein Merkmal)**.** Gegeben seien die Ausprägungen $(x_1, \ldots, x_n)$ eines Merkmals in n Spezies und eine Baumtopologie mit n Blättern, die mit $x_1, \ldots, x_n$ belegt sind, sowie eine Kostenfunktion, die einer Ausprägungsänderung von x nach x' die Kosten $\text{cost}(x, x') \geq 0$ zuordnet. Man spricht von *Einheitskosten*, wenn $\text{cost}(x, x') = [\![x \neq x']\!]$.

Gesucht ist eine Belegung der inneren Knoten und der Wurzel mit Merkmalsausprägungen, so dass die Kosten der Änderungen im Baum minimal sind. Die Kosten der Änderungen im Baum berechnen als Summe der Kosten aller Kanten, wobei die Kosten einer Kante die Kosten der Änderung von der Ausprägung im Startknoten zur Ausprägung im Zielknoten einer Kante sind.

Die minmalen Kosten der Änderungen im Baum bei optimaler Belegung heißen die Kosten der Baumtopologie für das betrachtete Merkmal. •

6.11 Problem (Großes Parsimony-Problem)**.** Gegeben sind die Ausprägungen $(x_i j)$, $1 \leq i \leq n$, $1 \leq j \leq m$ von m Merkmalen in n Spezies und eine Kostenfunktion cost_j für die Änderungen jedes Merkmals. Gesucht ist eine Baumtopologie mit minimalen Gesamtkosten. Die Gesamtkosten einer Baumtopologie sind die Summe der Kosten der Topologie über alle Merkmale. •

Wir bemerken, dass man das Große Parsimony-Problem theoretisch wie folgt lösen kann, wenn man das Kleine Parsimony-Problem lösen kann: Man berechnet für jede Baumtopologie die Kosten der Topologie für jedes Merkmal und somit die Gesamtkosten der Topologie und wählt dabei die optimale Baumtopologie aus. Unpraktisch ist dabei, dass man über superexponentiell viele Topologien iterieren muss. Mit einigen Tricks lässt sich aber das Große Parsimony-Problem in der Praxis für etwas mehr als 20 Taxa so lösen. Viel mehr darf man sich nicht erhoffen, denn es gilt der folgende Satz, dessen Beweis bei (**TODO: Ref**) nachgelesen werden kann.

6.12 Satz. *Das große Parsimony-Problem mit Einheitskosten ist NP-schwer.*

6.3.3 Das Kleine Parsimony-Problem

Sankoff's Dynamic Programming Algorithmus für das Kleine Parsimony-Problem. Wir leiten einen Dynamic Programming Algorithmus für das kleine Parsimony-Problem auf beliebigen gewurzelten Baumtopologien her. Wir gehen weiter unten auf den Fall eines ungewurzelten Baums ein.

Die Hauptidee ist, bei einer bottom-up-Durchmusterung für jeden Knoten v die optimalen Kosten $C(v, x)$ für den in v wurzelnden Unterbaum, wenn v mit der Ausprägung x belegt wird, zu berechnen.

In den Blättern ist die Belegung vorgegeben; daher setzen wir für ein Blatt v jeweils $C(v, x) := 0$, wenn x der Merkmalsausprägung in v entspricht, und $C(v, x) := \infty$ sonst. Für die inneren Knoten und die Wurzel gilt folgendes Lemma.

6.13 Lemma. Sei v ein innerer Knoten oder die Wurzel. Dann ist

$$C(v, x) = \sum_{c \text{ Kind von } v} \min_{x'} [C(c, x') + \text{cost}(x, x')]. \tag{6.1}$$

Beweis. Der Beweis erfolgt mit Induktion nach der (maximalen) Distanz von v zu einem Blatt. Für einen Knoten, der als Kinder nur Blätter hat, ist die Aussage klar. Sei nun das Lemma für alle Knoten unterhalb des betrachteten v bereits bewiesen. Sei x die Ausprägung in v, und sei x_c die Ausprägung im Kind c von v. Es ist klar, dass $C(v, x) \leq \sum_c [C(c, x_c) +$

$cost(x, x_c)$] für alle x_c gilt: Die optimalen Kosten in v mit x sind höchstens so hoch wie die Summe der optimalen Kosten in den Kindern plus die Kosten für die Kanten zu den Kindern. Wenn man die Belegung der Kinder so wählt, das jeder Summand minimal wird, erhält man $C(v, x) \leq \sum_{c \text{ Kind von } v} \min_{x'} [C(c, x') + \text{cost}(x, x')]$. Angenommen, es gilt $<$: Dann müsste es mindestens ein Kind c geben, in dem bei passender Ausprägung x' die Unterbaumkosten echt kleiner als $C(c, x')$ sind, was zum Widerspruch dazu steht, dass wir für c das Lemma schon bewiesen haben. Also gilt die Gleichheit. □

Wir definieren $B(c, x) := \text{argmin}_{x'} [C(c, x') + \text{cost}(x, x')]$ als die Belegung des Kindes c, die in (6.1) das Minimum annimmt.

Wenn wir bottom-up die Größen $C(v, x)$ für alle inneren Knoten (inkl. Wurzel) v und Ausprägungen x berechnet haben und $B(c, x)$ für alle inneren Knoten (ohne Wurzel) c und Ausprägungen x des Elternknoten kennen, dann wählen wir an der Wurzel r die Ausprägung $x_r := \text{argmin}_x C(r, x)$ und dann top-down im Kind c des schon belegten Knoten v die Ausprägung $x_c := B(c, x_v)$.

Laufzeit und Platzbedarf sind $\mathcal{O}(n\sigma^2)$, wobei σ die Anzahl der verschiedenen Ausprägungen des betrachteten Merkmals ist.

6.14 Beispiel (Kleines Parsimony-Problem)**.** Betrachte den Baum mit Ausprägungen

$$(((A,(G,G)),((T,T),A)),(T,(A,G))).$$

Durchführen des Algorithmus führt an der Wurzel zu A mit optimalen Kosten 4. ♡

Version von Fitch und Hartigan für Einheitskosten. Werden ausschließlich Einheitskosten verwendet, so lässt sich der obige Algorithmus anders darstellen; der Beweis der Äquivalenz ist eine Übungsausgabe. Jedem inneren Knoten v wird bottom-up eine Menge S_v zugeordnet: Einem Blatt wird die dort vorhandene Ausprägung zugeordnet. In einem inneren Knoten v setzen wir

$$S_v := \begin{cases} \bigcap_c S_c & \text{wenn } \bigcap_c S_c \text{ nicht leer,} \\ \bigcup_c S_c & \text{sonst.} \end{cases}$$

Schnitt und Vereinigung laufen dabei über alle Kinder c von v. Wir behaupten: S_v ist (bei Einheitskosten) die Menge der x, die $C(v, x)$ minimieren. Die endgültige Belegung wird top-down wie folgt durchgeführt: In der Wurzel r wähle irgendein $x_r \in S_r$. Für das Kind c eines schon mit x_v belegten Knoten v, wähle $x_c := x_v$ wenn $x_v \in S_c$, und irgendein $x_c \in S_c$ sonst.

Der Zeitaufwand für diese Variante von Fitch und Hartigan ist nur $\mathcal{O}(n\sigma)$.

Ungewurzelte Bäume. Ist der betrachtete Baum ungewurzelt, müssen zusätzlich annehmen, dass die Kostenfunktion eine Kostenmetrik ist (symmetrisch, Dreiecksungleichung).

Dann können wir die Wurzel auf irgend einer Kante platzieren und das gewurzelte Problem lösen. Wir behaupten: Die Lösung und die optimalen Kosten hängen nicht von der Platzierung der Wurzel ab. Denn: In welche Richtung die Kanten durchlaufen werden, spielt aufgrund der Symmetrie keine Rolle. Das Einfügen der Wurzel als zusätzlicher Knoten zwischen Knoten u, v kann die optimalen Kosten aufgrund der Dreiecksungleichung nicht verbessern.

Wählt man in der Wurzel aber die Merkmalsausprägung aus u oder v, werden die Kosten auch nicht verschlechtert.

Achtung: Wenn keine Kostenmetrik vorliegt, kann die optimale Lösung sehr wohl von der Platzierung der Wurzel abhängen.

6.3.4 Das Große Parsimony-Problem

Naiv löst man das Große Parsimony-Problem, in dem man alle Baumtopologien aufzählt, darin für jedes Merkmal die Kosten durch Lösen des Kleinen Parsimony-Problems berechnet und sich dabei die beste Topologie merkt.

Beschleunigung durch Branch-and-Bound. Die Auswertung einer einzelnen Topologie kann man (um einen geringen Faktor) beschleunigen, wenn man ausnutzt, dass mit jedem weiteren Merkmal die Kosten nur steigen können: Treten bei Merkmal j in der Matrix genau p verschiedene Ausprägungen auf (und sind Einheitskosten gegeben), dann erhöhen sich die Gesamtkosten mindestens um $p-1$. So kann man für alle noch nicht berechneten Merkmale abschätzen, welche Kosten *mindestens* noch hinzukommen, unabhängig von der Topologie.

Kennt man schon eine Topologie mit Gesamtkosten x und hat man für die aktuelle Topologie schon $m' < m$ Merkmale ausgewertet und kommt dabei bisher auf Kosten y' und weiß, dass noch mindestens Kosten y'' hinzukommen, und gilt $y := y' + y'' \geq x$, dann weiß man schon, ohne die verbleibenden Merkmale auszuwerten, dass die aktuelle Topologie nicht mehr besser als die beste bereits bekannte sein kann. Somit kann man die Auswertung vorzeitig abbrechen.

Deutlich besser ist folgende Idee, bei der man ganze Gruppen von Topologien aussparen kann. Organisiert man das Aufzählen aller Topologien wie in Abschnitt **??** angedeutet, also als Tiefensuche im Rekursionsbaum aller Topologien, dann werden durch Einfügen jedes Taxons ebenfalls die Kosten nur erhöht. Hat man mit den ersten n' Taxa in der aktuellen Teiltopologie schon Kosten y' erreicht und schätzt die minimalen noch hinzukommenden Kosten wieder geschickt mit y'' ab und kennt schon eine vollständige Topologie mit Gesamtkosten $x < y := y' + y''$, dann kann man das weitere Einfügen von Taxa abbrechen und den Suchraum hier abschneiden. Man spricht von "Branch-and-Bound".

Die Effektivität des Verfahrens hängt entscheidend davon ab, dass man schnell eine gute Topologie mit geringen Kosten x findet (durch eine Heuristik beispielsweise) und die Einfügereihenfolge der Taxa geschickt wählt. Mit diesem Verfahren ist das Große Parsimony-Problem in der Praxis für bis zu 25 Taxa (und nicht zu viele Merkmale) lösbar.

Man spricht hierbei von einer *Laufzeit-Heuristik*, da die Laufzeit bedeutend reduziert werden kann (aber nicht muss); in jedem Fall erhält man aber garantiert das optimale Ergebnis (wenn auch möglicherweise nach sehr langer Zeit).

Heuristiken. Um zu beginn eine Topologie mit (vielleicht nicht optimalen, aber hoffentlich) geringen Kosten zu finden, kann man irgendein (sinnvolles und schnelles) Verfahren einsetzen.

6.3.5 Inkonsistenz von Maximum Parsimony

Auch bei vielen Merkmalen, die alle auf die gleiche (korrekte) Topologie hindeuten, kann es vorkommen, dass Maximum Parsimony den "falschen" Baum berechnet. Long Branch Attraction. (Übungsaufgabe) (**TODO: ausarbeiten**)

6.4 Minimum-Flip-Probleme

Wie bereits erwähnt, besteht eine andere Sichtweise auf (binäre) Merkmalsdaten, die keine perfekte Phylogenie bilden, darin, zu fragen, welches die "nächstgelegene" Datenmatrix ist, die eine perfekte Phylogenie zulässt. Die Flip-Distanz zwischen zwei Matrizen ist dann einfach die Anzahl der elementweisen Unterschiede.

Zusätzlich kann man hierbei zulassen, dass in den Ausgangsdaten einige der Merkmalsausprägungen unbekannt sind; dies kennzeichnen wir mit einem Fragezeichen. Das Ändern eines Fragezeichens in 0 oder 1 kostet nichts (trägt nicht zur Distanz bei). Die Flip-Distanz ist dadurch weder eine Metrik noch eine Pseudo-Metrik.

6.15 Definition (Flip-Distanz)**.** Die *Flip-Distanz* $d(W, W')$ zwischen zwei Matrizen $W, W' \in \{0, 1, ?\}^{n \times m}$ ist die Anzahl der Elemente die in der einen Matrix 0 und in der anderen 1 sind.

6.16 Definition (PP-Matrix)**.** Eine *PP-Matrix* ist eine Matrix $X \in \{\,0, 1\,\}^{n \times m}$, zu der eine perfekte Phylogenie mit Zustand 0^m in der Wurzel existiert.

Insbesondere darf eine PP-Matrix keine Fragezeichen enthalten.

Wir erhalten nun zwei Problemvarianten, je nachdem, ob in der Ausgangsmatrix Fragezeichen zugelassen sind oder nicht. Beide Varianten lassen sich aber im Folgenden exakt gleich behandeln. Die Probleme heißen Minimum Flip Consensus Tree (MFCT, keine Fragezeichen zugelassen) und Minimum Flip Supertree (MFST, Fragezeichen zugelassen). Die Namen kommen daher, dass eine Hauptanwendung dieser Fragestellung darin liegt, zu bereits vorhandenen Bäumen, die sich teilweise widersprechen können, einen Konsensus-Baum bzw. Oberbaum (hier sind einige Taxa in einigen Bäumen nicht enthalten) zu berechnen. Die Merkmale spiegeln dabei die Existenz aller Unterbäume (Splits) wider.

6.17 Problem (MFCT)**.** Gegeben $W \in \{0, 1\}^{n \times m}$, finde eine PP-Matrix X, die $d(W, X)$ minimiert. •

6.18 Problem (MFST)**.** Gegeben $W \in \{0, 1, ?\}^{n \times m}$, finde eine PP-Matrix X, die $d(W, X)$ minimiert. •

Beide Probleme sind NP-schwer **?**. Daher versuchen wir hier eine Lösung mit Hilfe von ganzzahligen linearen Programmen, die in der Praxis noch bis zu ca. 100 Taxa und mehreren hundert Merkmalen funktioniert.

Zur Abkürzung seien $\mathcal{T} := \{1, \ldots, m\}$ und $\mathcal{C} := \{1, \ldots, n\}$ die Indexmengen der Taxa (Zeilen) und Merkmale (Spalten) der Eingabematrix $W \in \{0, 1, ?\}^{m \times n}$.

Die natürlichen Variablen des Problems sind durch ein $n \times m$-Matrix

$$X = (x_{tc})_{t\in\mathcal{T},c\in\mathcal{C}} \quad \in \{0,1\}^{n\times m} \tag{6.2}$$

gegeben, die eine Lösung darstellt mit $x_{tc} = 1$ genau dann wenn in Taxon t das Merkmal c existiert.

Um die Zielfunktion $d(W, X)$ als lineare Funktion der x_{tc} darzustellen, definieren wir Hilfskonstanten

$$\omega_{tc} := \begin{cases} 1 & \text{if } W_{t,c} = 0, \\ 0 & \text{if } W_{t,c} = ?, \\ -1 & \text{if } W_{t,c} = 1, \end{cases} \qquad \text{für } t \in \mathcal{T}, c \in \mathcal{C}. \tag{6.3}$$

Damit ist, wie man schnell nachrechnet,

$$d(W, X) = \text{Ones}(W) + \sum_{t\in\mathcal{T},c\in\mathcal{C}} w_{tc} \cdot x_{tc}, \tag{6.4}$$

wobei $\text{Ones}(W)$ die Anzahl der Einsen in W ist.

Jetzt müssen wir noch die Menge der zulässigen X, also der PP-Matrizen, mit Hilfe linearer Ungleichungen beschreiben. Hierfür gibt es mehrere Möglichkeiten: Wir können beispielsweise die PP-Bedingungen aus Satz 6.7 zunächst so umschreiben: Für alle Paare con Spalten $x_{\cdot,c}$ und $x_{\cdot,c'}$ mit $c < c'$ gilt:

$$xcolc \leq x_{\cdot,c'} \text{ oder } x_{\cdot,c'} \leq x_{\cdot,c} \text{ oder } x_{\cdot,c'} + x_{\cdot,c} \leq \mathbf{1};$$

dabei ist $\mathbf{1}$ der Vektor aus lauter Einsen. Jetzt können wir weitere Variablen für jedes Paar von Spalten einführen, um sicherzu stellen, dass jeweils mindestens eine dieser Ungleichungen erfüllt ist (Übungsaufgabe).

Hier wählen wir einen Weg, der ohne weitere Variablen auskommt. Sei $G(X) := (V, E)$ der ungerichtete bipartite Graph zu X mit $V = \mathcal{T} \cup \mathcal{C}$ und $E = \{\, \{\, t, c \,\} : X_{tc} = 1 \,\}$. Betrachten wir den von beliebigen $c_1 < c_2$ und paarweise verschiedenen t_1, t_2, t_3 induzierten Teilgraph (bzw. den entsprechenden Ausschnitt aus der X-Matrix), so nennen wir diesen ein "M", wenn folgende Situation vorliegt:

	c_1	c_2
t_1	1	0
t_2	1	1
t_3	0	1

Die Benennung ist klar, wenn man die c-Knoten oben und die t-Knoten unten hinschreibt. Aus Satz 6.7 wissen wir, dass X genau dann eine PP-Matrix ist, wenn diese Situation nicht auftritt, wenn also $G(X)$ M-frei ist.

Jetzt müssen wir dies nur noch mit linearen Ungleichungen aufschreiben: Um das dargestellte "M" zu vermeiden, dürfen entweder nur drei der vier Katen $(t_1, c_1), (t_2, c_1), (t_2, c_2), (t_3, c_2)$ vorhanden sein, oder mindestens eine der beiden anderen Kanten muss vorhanden sein. Dies führt auf die "M"-Ungleichungen

$$\begin{aligned} &x_{t_1,c_1} + x_{t_2,c_1} + x_{t_2,c_2} + x_{t_3,c_2} \leq 3 + x_{t_1,c_2} + x_{t_3,c_1} \\ &\forall\ t_1, t_2, t_3 \in \mathcal{T} \text{ (paarweise verschieden)}, \quad \forall\ c_1, c_2 \in \mathcal{C},\ c_1 < c_2. \end{aligned} \tag{6.5}$$

Insgesamt haben wir dank 6.7 folgende Aussage.

6.19 Satz. *Die zulässige Lösungen des ILPs*

$$\min (6.4), \text{ so dass } (6.2), (6.5)$$

sind genau die PP-Matrizen X, und die lineare Funktion (6.4) *stellt die Kosten der Lösung X dar.*

Die Anzahl der Variablen ist in dieser Formulierung minimal (nur die nm natürlichen Variablen treten auf). Die Anzahl der Ungleichungen ist jedoch mit $n(n-1)(n-2)\binom{m}{2} = \Theta(n^3m^2)$ sehr hoch.

Lösung des ILPs. Die Lösung eines solchen ganzzahligen linearen Programms (integer linear program, ILP) geschieht mit folgender Strategie:

1. Löse die zugehörige LP-Relaxierung (d.h. das Problem ohne die Ganzzahligkeitsbedingung; $x_{tc} \in \{0, 1\}$ wird aufgeweicht zu $0 \leq x_{tc} \leq 1$).
2. Ist die Lösung ganzzahlig, so wurde ein zulässiges X gefunden und kann mit der bisher bekannten besten zulässigen Lösung verglichen werden.
3. Ist die Lösung nicht ganzzahlig, gibt es mehrere Möglichkeiten: Man kann in zwei Unterprobleme verzweigen, indem man ein fraktionales x_{tc} auswählt und dies einmal auf 1 und einmal auf 0 festsetzt (branch), oder man kann eine Hyperebene (cut) finden, die die gefundene fraktionale Lösung von der zulässigen Menge trennt, oder dies kombinieren (branch-and-cut). Bei dem hier vorliegenden binären Problem ist es am einfachsten, in beide Fälle zu verzweigen und zu beachten, dass der optimale Zielfunktionswert gazzahlig ist.

Da jedoch die Anzahl der Ungleichungen zu groß ist, um die Koeffizientenmatrix des LP nur aufzustellen, muss man mit weiteren Tricks arbeiten: Man lässt zunächst alle "M"-Ungleichungen weg und löst das LP ohne diese. Natürlich erhält man so oft eine Lösung, die fraktional ist und viele der Ungleichungen nicht erfüllt; es kann aber passieren, dass alle Ungleichungen glücklicherweise schon erfüllt sind. Man muss also testen, ob es mindestens eine unerfüllte "M"-Ungleichung gibt. Wenn das der Fall ist, fügt man diese (oder eine kleine Menge von solchen) dem LP hinzu und löst erneut. Die Hoffnung ist, dass man, nachdem man einige wenige entscheidende Ungleichungen hinzugefügt hat, die meisten anderen von selbst erfüllt sind, wenn die Zielfunktion optimiert wird. Beim Minimum Flip Problem ist es sinnvoll, für alle paare von Spalten die $c_1 < c_2$ die drei Zeilen zu finden, die die "M"-Ungleichung am stärksten verletzen, also

$$\begin{aligned} &x_{t_1,c_1} + x_{t_2,c_1} + x_{t_2,c_2} + x_{t_3,c_2} - x_{t_1,c_2} - x_{t_3,c_1} \\ =&(x_{t_1,c_1} - x_{t_1,c_2}) + (x_{t_2,c_1} + x_{t_2,c_2}) + (x_{t_3,c_2} - x_{t_3,c_1}) \to \max . \end{aligned}$$

Dies wird offensichtlich dadurch erreicht, dass man

$$\begin{aligned} t_1 &:= \operatorname*{argmax}_t (x_{t,c_1} - x_{t,c_2}) \\ t_2 &:= \operatorname*{argmax}_t (x_{t,c_1} + x_{t,c_2}) \\ t_3 &:= \operatorname*{argmax}_t (x_{t,c_2} - x_{t,c_1}) \end{aligned}$$

findet, was problemlos in $O(n)$ Zeit möglich ist. Man führt dies für alle Spaltenpaare in $O(nm^2)$ Zeit insgesamt durch und fügt alle verletzten Ungleichungen zum LP hinzu.

Ein weiteres Problem ist die große Zahl an Variablen. Hier kann man einen weiteren Trick anwenden: Man codiert die Lösung nicht durch die natürlichen Variablen x_{tc}, sondern anhand der Differenz zu den Daten. Dabei versucht man zunächst die korrekten Werte für die ?-Einträge vorab geschickt zu erraten, so dass keine ?-Einträge mehr vorhanden sind. Die falsch geratenen Einträge können immer noch kostenfrei geändert werden. Ist $W_{tc} = 0$, verwenden wir wie gehabt x_{tc}, benötigen die Variable aber nicht, solange sie den Wert Null annimmt. Ist $W_{tc} = 1$, verwenden wir nicht x_{tc}, sondern die komplementäre Variable $y_{tc} := 1 - x_{tc}$ und ersetzen in den Ungleichungen und in der Zielfunktion x_{tc} durch $1 - y_{tc}$ für diejenigen (t, c) mit $W_{tc} = 1$. Weichen W und die optimale Lösung X an nur wenigen Stellen voneinander ab, sind fast alle Variablen Null. Variablen, die Null sind und nie betrachtet werden, müssen nicht in die Problemformulierung aufgenommen werden. Wir fügen eine Variable nur dann zum Problem hinzu, wenn dies z.B. durch Hinzufügen einer verletzen Ungleichung notwendig wird. Die Art der Variablen (ob x_{tc} oder $y_{tc} = 1 - x_{tc}$) verwendet wird, kann im Prinzip in jedem Knoten neu entschieden werden; insbesondere für die ?-Einträge.

Für die eingentliche Lösung des LPs verwendet man einen Standard-LP-Solver, am besten mit einem Front-End, dass die Formulierung und Lösung des Separierungsproblems und der Column Generation auf einfache Art und Weise ermöglicht. Hierzu gibt es zahlreiche kommerzielle und Open-Source-Tools.

KAPITEL 7

Phylogenetik: Distanzbasierte Methoden

7.1 Metriken

Bisher haben wir merkmalsbasierte Methoden in der Phylogenetik betrachtet, bei denen eine $n \times m$-Merkmalsausprägungsmatrix gegeben ist. In diesem Abschnitt gehen wir davon aus, dass eine Distanzmatrix $D = (D_{ij})$ gegeben ist, wobei D_{ij} den evolutionären Abstand zwischen Taxa i und j darstellt. In diesem Fall kann man die Kantenlänge in einem Baum proportional zur Distanz zwischen zwei Knoten darstellen. Woher man solche Distanzen bekommt, lassen wir im Moment beiseite, und gehen davon aus, dass eine passende Matrix D gegeben ist.

Generell geht es nun darum, eine (ungewurzelte) Baumtopologie $T = (V, E)$ und Kantenlängen $\ell : E \to \mathbb{R}_{\geq 0}$ zu finden, so dass die Summe der Kantenlängen auf dem Pfad zwischen zwei Blättern i, j "möglichst gut" die gegebene Distanz D_{ij} abbildet. Es gibt auch die Variante, einen gewurzelten Baum zu berechnen, in dem zusätzlich jeder Pfad von der Wurzel bis zu den Blättern gleich lang ist. Ein solcher Baum heißt *ultrametrisch*.

Eine exakte Lösung wird im Normalfall für beide Varianten nicht möglich sein, da ja (bei Symmetrie der Distanzen) $n(n-1)/2 = \Theta(n^2)$ Werte gegeben sind, ein ungewurzelter Binärbaum mit n Blättern aber nur $2n - 3 = \Theta(n)$ Katen hat, deren Längen man bestimmen kann. Wir geben nun den speziellen Distanzmatrizen, die eine exakte Lösung des Problems ermöglichen, spezielle Namen und widerholen zunächst den Begriff der Metrix. (Wir setzen immer voraus, dass D eine Metrik ist.)

7.1 Definition (Metrik, additive Metrik, Ultrametrik)**.** Sei $D \geq 0$ eine $n \times n$-Distanzmatrix. Dann heißt D eine *Metrik* genau dann wenn

1. $D_{ij} = 0$ genau dann wenn $i = j$ (Definitheit),

2. $D_{ij} = D_{ji}$ für alle i, j (Symmetrie),
3. $D_{ij} \leq D_{ik} + D_{kj}$ für alle i, j, k (Dreiecksungleichung).

Eine Metrik D heißt *additive Metrik* genau dann wenn es einen ungewurzelten Baum mit Kantenlängen gibt, in dem die Pfadlängen zwischen den Blättern mit den gegebenen Distanzen übereinstimmen.

Eine Metrik D heißt *Ultrametrik* genau dann wenn es einen gewurzelten Baum mit Kantenlängen gibt, in dem die Pfadlängen zwischen den Blättern mit den gegebenen Distanzen übereinstimmen und alle Pfade von der Wurzel zu den Blättern gleich lang sind.

Natürlich ist die Definition über die Existenz eines Baumes nicht hilfreich, aber glücklicherweise gibt es eine davon unabhängige Charakterisierung.

7.2 Lemma (additive Metrik, Ultrametrik)**.** Eine Metrik ist genau dann additiv, wenn zu je vier Objekten U, V, X, Y von den drei Distanzsummen $S_1 := D(U, V) + D(X, Y)$, $S_2 := D(U, X) + D(V, Y)$ und $S_3 := D(U, Y) + D(V, X)$ die beiden größten gleich sind (sogenannte *Vier-Punkte-Bedingung*).

Eine Metrik ist genau dann eine Ultrametrik, wenn zu je drei Objekten U, V, X von den drei Distanzen $D(U, V)$, $D(U, X)$ und (V, U) die beiden größten gleich sind (*Drei-Punkte-Bedingung*).

Beweis. Durch eine Skizze und Ausrechnen der drei Distanzsummen in Quartetten bzw. Betrachten der drei Distanzen in einem gewurzelten Tripel wird klar, dass in einem ungewurzelten bzw. gewurzelten ultrametrischen Baum die Pfadlängen die jeweilige genannte Bedingung erfüllen.

Der Beweis der Umkehrung erfolgt konstruktiv durch Angabe von zwei Algorithmen in den folgenden Abschnitten. Der erste konstruiert zu jeder Distanzmatrix einen ultrametrischen Baum (und damit auch ein hierarchisches Clustering der Objekte). Wir werden zeigen, dass für eine Metrik D, die die Drei-Punkte-Bedingung erfüllt, die Pfadlängen des Baums den gegebenen Abständen entsprechen. Der zweite Algorithmus (NJ) konstruiert zu jeder Distanzmatrix einen ungewurzelten Baum mit Kantenlängen. Auch hier werden wir sehen, dass im Fall einer Metrik D, die die 4-Punkte-Bedingung erfüllt, die Pfadlängen im Baum den gegebenen Abständen entsprechen. □

7.2 Agglomeratives Clustern

Die Idee beim agglomerativen Clustern ist, in einer Distanzmatrix $D = (D_{ij})$

1. das Paar $i \neq j$ mit dem minimalen Abstand zu finden
2. i und j zu einem neuen Objekt (Unterbaum) $\{ i, j \}$ zu verschmelzen und die Zeilen und Spalten i und j aus D zu streichen,
3. in D eine neue Zeile und Spalte $\{ i, j \}$ einzufügen, die die Abstände $D(i, k)$ und $D(j, k)$ für alle verbleibenden $k \neq i, j$ zu einem "Mittelwert" $D(\{ i, j \}, k)$ kombiniert,

4. die Schritte 1.–3. solange zu wiederholen, bis nur noch ein einzelnes Objekt übrig ist, wobei sich in jeder Iteration die Anzahl der verbleibenden Objekte um eins verringert.

Zwei Details sind zu klären:

- Auf welcher Höhe liegt die Wurzel des Unterbaums $\{ i, j \}$? Da wir einen Ultrametrischen Baum konstruieren wollen, bleibt nur die Möglichkeit $D(i, j)/2$. Da wir (per Induktion) wissen, dass die Wurzeln der Unterbäume (bzw. die Blätter) i und j tiefer (oder zumindest nicht höher) liegen, ergibt sich eine nichtnegative Kantenlänge von den Wurzeln von i und j zur Wurzel von $\{ i, j \}$.
- Wie berechnet sich in Schritt 3. der "Mittelwert" für das neue Objekt $z := \{ i, j \}$? In der Tat gibt es verschiedene Varianten des Agglomerativen Clusterns, die sich exakt hierin unterscheiden, u.a.
  1. Single-linkage Clustering: $D(z, k) := \min \{ D(i, k), D(j, k) \}$.
  2. Complete-linkage Clustering: $D(z, k) := \max \{ D(i, k), D(j, k) \}$.
  3. Weighted Pair Group Method using Averages (WPGMA): $D(z, k) := [D(i, k) + D(j, k)]/2$.
  4. Unweighted Pair Group Method using Averages (UPGMA): $D(z, k) := [S_i D(i, k) + S_j D(j, k)]/(S_i + S_j)$, wobei S_i und S_j die jeweils Anzahl der Objkete im Cluster i und j sind (1 für ein Blatt).

  In jedem Fall liegt $D(z, k)$ zwischen $D(i, k)$ und $D(j, k)$.

7.3 Satz (Agglomeratives Clustern bei Ultrametriken)**.** *Erfüllt D die Drei-Punkte-Bedingung, dann sind die vorgestellten Varianten äquivalent, der nach einem Iterationsschritt erzeugte Baum z ist ein ultrametrischer Baum, in dem die Pfadlängen zwischen den Blättern mit den gegebenen Distanzen übereinstimmen. und die reduzierte Datenmetrix D ist erfüllt wieder die Drei-Punkte-Bedingung.*

Beweis. Die vier genannten (und weitere sinnvolle) Varianten sind äquivalent, weil (i, j) ein Paar mit minimaler Distanz ist, so dass $D(i, k), D(j, k) \geq D(i, j)$ gilt. Aufgrund der Drei-Punkt-Bedingung gilt dann $D(i, k) = D(j, k) = D(z, k)$ für jede der genannten Kombinationsvarianten. (Auf diese Art und Weise kann man während des Ablaufs des Algorithmus prüfen, ob die Drei-Punkte-Bedingung erfüllt ist!)

Nach Induktion sind i und j selbst bereits ultrametrische Bäume (oder Blätter), die nun zu einem neuen Baum mit Wurzel in Höhe $D(i, j)/2$ kombiniert werden. Wir haben bereits gesehen, dass die Kanten zu den Wurzeln von i und j nichtnegative Länge haben. Damit haben wir einen ultrametrischen Baum $z = \{ i, j \}$, in dem die Pfade von der Wurzel zu jedem Blatt die Länge $D(i, j)/2$ haben. Von den Unterbäumen i und j wussten wir schon, dass sie ultrametrisch sind und die Pfadlängen die Distanzen korrekt abbilden. Jetzt müssen wir nur noch die Pfade zwischen je einem Blatt i' in i und einem Blatt j' in j betrachten: Diese haben nun alle die Länge $D(i, j)$, und aufgrund der Drei-Punkt-Eigenschaft ist klar, dass in der ursprünglichen Matrix D alle die gleiche Länge gehabt haben müssen.

Nach der Reduktion entspricht $D(z, k)$ den Werten $D(i, k) = D(j, k)$. Die Matrix hat sich also nicht verändert; es wurde nur die i-te (oder j-te) Zeile und Spalte gestrichen. Galt die Drei-Punkt-Bedingung vorher, gilt sie immer noch. □

Laufzeit. Die Laufzeit des Verfahrens beträgt zunächst $O(n^3)$, da es $n-1$ Iterationen gibt, in denen das Minimum der Matrix gefunden werden muss. Mit Hilfe von Quadtrees (Epstein, Fast hierachical clustering and other applications of dynamic closest pairs, Journal of Experimental Algorithms 5:1–23, 2000) lässt sich die Laufzeit auf $O(n^2)$ reduzieren. Quadtrees erlauben das Einfügen und Löschen eines Objekts in $O(n)$ Zeit und ebenso das Auffinden eines Paars mit minimalem Abstand.

Andere Ideen zur Verbesserung der Laufzeit, für die man keine Quadtrees benötigt, lauten wie folgt: Man ermittelt zunächst anfangs zu jedem Objekt i den nächsten Nachbarn

$$N_i := \operatorname*{argmin}_{j \neq i} \{ D_{ij} \} .$$

(Im Allgemeinen folgt aus $N_i = j$ nicht notwendigerweise $N_j = i$!) Dafür benötigt man einmalig $O(n^2)$ Zeit. Um ein Paar mit minmalem Abstand zu finden, muss man nun nur noch die Paare (i, N_i) für $i = 1, \ldots, n$ betrachten. Die Kunst ist, nach einer Verschmelzung die (nötigen) N_i schnell neu zu berechnen.

Im Fall des Single-linkage Clusterings ist das aber einfach effizient möglich: Werden i und j zu $z = \{ i, j \}$ verschmolzen und ist für ein k bisher $N_k = i$ oder $N_k = j$, dann gilt danach $N_k = z$, und die anderen N_k ändern sich nicht. Damit werden alle N_k für $k \neq z$ jeweils in konstanter Zeit, also insgesamt in $O(n)$ Zeit aktualisiert. Der neue Wert N_z kann trivial in $O(n)$ Zeit gefunden werden. Insgesamt benötigt man so nur $O(n)$ Zeit pro Iteration und insgesamt also $O(n^2)$.

Da bei ultrametrischen Daten alle vier genannten Varianten das gleiche Ergebnis liefern, nämlich den korrekten ultrametrischen Baum, lässt sich also in $O(n^2)$ selbiger aus ultrametrischen Daten finden. Insbesondere lässt sich (vgl. die Bemerkung im Beweis von Satz 7.3) in $O(n^2)$ Zeit prüfen, ob eine Metrik eine Ultrametrik ist.

Einfache Methoden, um auch UPGMA und WPGMA in $O(n^2)$ ohne komplexe Datenstrukturen auf beliebigen Eingaben durchzuführen, werden von Gronau und Moran (Optimal Implementations of UPGMA and Other Common Clustering Algorithms, Information Processing Letters 104(6):205–210, 2007) beschrieben, zusammen mit einer Charakterisierung, bei welchen Kombinationsregeln eine geschickte Nachbar-basierte Suche genau äquivalent zur Minimum-Suche über alle $O(n^2)$ Matrixelemente ist. Ein weiteres Beispiel für eine ähnliche Beschleunigung lernen wir im nächsten Abschnitt kennen.

Zusammenfassung. Die hier genannten Methoden berechnen in jedem Fall einen ultrametrischen Baum. Die Längen der Pfade zwischen den Blättern stimmen genau dann mit den vorgegebenen Distanzen überein, wenn die Distanzmatrix eine Ultrametrik war. Ansonsten bekommt man einen Baum, der von der Art der Kombinationsregel abhängt und eventuell vom verwendeten Algorithmus, der das nächste zusammenzufassende Blattpaar auswählt. Ein so berechnete Baum erfüllt insbesondere *kein* offensichtliches Optimalitätskriterium (etwa, dass die Summe der Abweichungen zu den vorgegebenen Distanzen minimiert wird), und er kann einfach falsch sein, und zwar schon dann, wenn die Eingabe eine additive Metrik ist. Dies überlegt man sich am Minimalbeispiel des Quartetts (A:1, B:3 ‖ C:1, D:3), wobei die Mittelkante die Länge 1 haben soll.

7.3 Neighbor Joining auf additiven Distanzen

7.3.1 Motivation

Ein einfaches Beispiel zeigt, dass UPGMA bei einer nicht ultrametrischen aber additiven Metrik nicht immer den korrekten Baum konstruiert: Betrachte das Quartett $(ab : cd)$, bei dem die Blattkanten zu a und c die Länge 1, die Blattkanten zu b und d die Länge 3 haben und die innere Kante die Länge 1 hat. Im ersten UPGMA-Clustering-Schritt würde fälschlich das Paar $\{a, c\}$ zusammengefasst, da es die geringste Distanz (3) hat.

Im Folgenden stellen wir zunächst den *Neighbor Joining* Algorithmus vor, der auf additiven Metriken einen ungewurzelten Baum konstruiert, in dem die Distanzen zwischen allen Blatt-Paaren den gegebenen Distanzen entsprechen.

Wir besprechen dann zwei Verbesserungen (man lasse sich dabei von der unterschiedlichen Bedeutung des Wortes "fast" im Deutschen und im Englischen nicht verwirren!):

1. NJ rekonstruiert auch dann den korrekten Baum, wenn die Eingabe-Distanzen nur *fast additiv* sind. Der Beweis dieser Tatsache ist nur wenig aufwändiger als der Beweis für additive Distanzen selbst.
2. In seiner Rohform benötigt NJ auf n Taxa $O(n^3)$ Zeit. Dies lässt sich jedoch auf $O(n^2)$ verbessern; den resultierenden Algorithmus nennt man *Fast Neighbor Joining* (FNJ). Die resultierenden Bäume sind bei additiven und fast additiven Eingabe-Metriken die gleichen; auf anderen Eingaben können die Ergebnisse abweichen.

7.3.2 Definitionen

Wir definieren zunächst den Nachbarschaftsbegriff und Nachbarquartette.

7.4 Definition (Nachbarn, Nachbarpaar)**.** Zwei Blätter a, b heißen *Nachbarn* oder *Nachbarpaar* in einem Baum T, wenn der Pfad zwischen ihnen aus genau zwei Kanten besteht.

In einem Binärbaum hat ein Blatt höchstens einen Nachbarn, aber nicht jedes Blatt muss einen Nachbarn haben.

7.5 Definition (Nachbarquartett)**.** Sei T ein Baum und $\{i, j\}$, $\{k, l\}$ zwei verschiedene Nachbarpaare in T. Dann sagen wir, T enthält das *Nachbarquartett* $(ij : kl)$.

7.6 Lemma (Abstand zwischen Elternknoten von Nachbarpaaren)**.** Sei D die von T induzierte additive Metrik. Seien $\{i, j\}$, $\{k, l\}$ zwei verschiedene Nachbarpaare in T mit Eltern $\overline{ij}$ und $\overline{kl}$. Dann kann man den Abstand zwischen $\overline{ij}$ und $\overline{kl}$ berechnen als

$$w_D(ij : kl) := (D(i,k) + D(i,l) + D(j,k) + D(j,l) - 2D(i,j) - 2D(k,l))/4.$$

Beweis. Durch Zeichnen des Nachbarquartetts $(ij : kl)$ und Abzählen, wie oft jede Kante benutzt wird. □

Algorithm 2 NJ(D): berechnet aus der Metrik D auf der Objektmenge $N(D)$ mit $|N(D)| \geq 3$ einen ungewurzelten Baum

$D_1 \leftarrow D$
for $i \leftarrow 1, \ldots, |N(D)| - 3$ **do**
  $\{a_i, b_i\} \leftarrow \text{argmin}_{x \neq y} S_{D_i}(x, y)$
  Reduziere das Paar $\{a_i, b_i\}$ auf einen neuen Knoten c_i;
    berechne Kantenlängen von $\{a_i, c_i\}$ und $\{b_i, c_i\}$ (Details später).
  Berechne $D' :\equiv D_{i+1}$ auf $N(D') := N(D) \setminus \{a_i, b_i\} \cup \{c_i\}$:
    für $x, y \neq c_i$ ist $D'(x, y) = D_i(x, y)$ unverändert;
    sonst berechne $D'(x, c_i)$ aus $D_i(x, a_i)$ und $D_i(x, b_i)$ (Details später).
Verbinde die 3 verbleibenden Objekte zu einem Sternbaum mit passenden Kantenlängen.
return Zentrum des Sternbaums.

7.3.3 Übersicht und Algorithmus

Die Idee ist grundsätzlich die gleiche wie bei den hierarchischen Clustering-Algorithmen UPGMA, etc.: Zu einer gegebenen Metrik D wählen wir ein geeignetes Paar an Objekten (Taxa) aus, fassen diese zu einem neuen Objekt zusammen, entfernen die ursprünglichen Objekte aus der Menge der zu bearbeitenden Objekte und fügen das neue hinzu, nachdem wir die Distanzen zwischen dem neuen und allen verbleibenden Objekten berechnet haben. Dadurch wird in jedem Schritt die Menge der zu bearbeitenden Objekte um ein Objekt reduziert, bis nur noch drei Objekte übrig sind. Diese werden zu einem Sternbaum zusammengefasst.

Der Unterschied zwischen UPGMA und NJ besteht in der Berechnung des Auswahlkriteriums für die zusammenzufassenden Objekte. Bei UPGMA wurde das Paar mit minimaler Distanz ausgewählt, was bei Ultrametriken aufgrund der 3-Punkt-Bedingung zu korrekten Ergebnissen führt, bei additiven Metriken jedoch nicht immer (siehe Beispiel oben).

Die Struktur des Algorithmus ist in Algorithmus 2 beschrieben. Aus der Eingabe D mit Objektmenge $N(D)$ mit $|N(D)| = n \geq 3$ wird eine Folge $D = D_1, D_2, \ldots D_{n-3}$ von reduzierten Metriken konstruiert. Wir lassen dabei momentan noch Details offen (etwa, wie die neuen Distanzen oder die Kantenlängen der zusammengesetzten Objekte genau gewählt werden).

In der angegebenen Form ergibt sich mit $|N(D)| = n$ eine Laufzeit von $O(n^3)$. Wir werden sehen, dass sich dies noch verbessern lässt.

7.3.4 Das Auswahlkriterium S_D

Gedanken zur Identifizierung eines benachbarten Paars. Wenn $\{a, b\}$ in einem Baum T mit additiver Distanz D ein benachbartes Paar ist, dann existiert für alle $c, d \notin \{a, b\}$ das Quartett $(ab : cd)$ in T (es ist nicht unbedingt ein Nachbarquartett, da c und d nicht notwendigerweise Nachbarn sind). Insbesondere ist dann $w_D(ab : cd)$ für alle c, d nichtnegativ. Sind aber a, b nicht benachbart, dann kann es Belegungen für c, d geben, für die $w_D(ab : cd)$ negativ wird.

Dies legt nahe, die Summe

$$Z_D(a,b) := \sum_{x,y \notin \{a,b\}} w_D(ab:xy)$$

zu maximieren, also $\{a,b\} = \operatorname{argmax}_{u,v} Z_D(u,v)$ als Nachbarpaar auszuwählen.

In der Tat ist dies das korrekte Kriterium; es lässt sich jedoch erheblich vereinfachen. Durch Ausrechnen kann man zeigen, dass

$$Z_D(a,b) = -G - (n-1)\cdot((n-2)D(a,b) - R(a) - R(b)),$$

wobei $R(x) := \sum_y D(x,y)$ und $G := \sum_x R(x) = \sum_{x,y} D(x,y)$ gesetzt wurde. Da G nicht von a,b abhängt, ist die Maximierung von $Z_D(a,b)$ äquivalent zur Minimierung von

$$S_D(a,b) := (n-2)\cdot D(a,b) - R(a) - R(b). \tag{7.1}$$

Im Auswahlschritt des NJ-Algorithmus wird also das Paar $\{a,b\}$ gefunden, dass S_D minimiert. Im Folgenden beweisen wir einige Eigenschaften des Auswahlkriteriums S_D, die dessen Wahl rechtfertigen.

Eigenschaften des Auswahlkriteriums. Wir erwähnen zunächst eine Verschiebungsinvarianz von S_D, dessen Beweis durch Nachrechnen erfolgt.

7.7 Lemma (Verschiebungslemma; Mihaescu et al., 2006)**.** Sei $\tilde{D}$ eine Verschiebung von D, d.h., es gebe ein Blatt $c \in N(D)$ und eine Konstante σ, so dass

$$\begin{aligned}
\tilde{D}(x,c) &= D(x,c) + \sigma \text{ für } x \neq c,\\
\tilde{D}(c,c) &= D(c,c) = 0,\\
\tilde{D}(x,y) &= D(x,y) \text{ für alle } x,y \neq c.
\end{aligned}$$

Dann ist $S_{\tilde{D}}(x,y) = S_D(x,y) - 2\sigma$ für alle x,y.

Wir kommen jetzt zu den *wichtigen* Eigenschaften von S_D, die im folgenden Lemma und Satz zusammengefasst sind.

7.8 Lemma (Saitou & Nei, 1987)**.** Sei T ein ungewurzelter Binärbaum mit gegebenen Kantenlängen, D die zugehörige additive Metrik, und S_D wie in (7.1) definiert. Seien a,b Nachbarn in T. Dann ist $S_D(a,b) = \min_{y \neq a} S_D(a,y)$.

Beweis. Sei $y \neq a,b$. Nach Definition ist

$$\begin{aligned}
S_D(a,y) - S_D(a,b) &= (n-2)\cdot[D(a,y) - D(a,b)] + R(b) - R(y)\\
&= \sum_{x \neq a,b,y} [D(a,y) - D(a,b) + D(x,b) - D(x,y)]\\
&\geq 0,
\end{aligned}$$

denn nach der 4-Punkt-Bedingung ist stets $D(a,b)+D(x,y) \leq D(a,y)+D(x,b)$, also für jedes $x \neq a,b,y$ der Summand nichtnegativ. Damit ist $S_D(a,b) \leq S_D(a,y)$ für alle $y \neq a,b$. □

Das Lemma sagt aus, dass der Nachbar b von a (wenn es ihn gibt), die Funktion $S_D(a,\cdot)$ minimiert. Unklar ist noch, dass das Minimum über beide Argumente tatsächlich ein Nachbarpaar liefert, denn nicht jedes Blatt muss Nachbarn haben. Wichtiger ist daher die folgende Umkehrung des Lemmas.

7.9 Satz (Studier & Keppler, 1988)**.** *Sei T ein ungewurzelter Binärbaum mit gegebenen Kantenlängen, D die zugehörige additive Metrik, und S_D wie in* (7.1) *definiert. Sei* $\{a,b\} := \operatorname{argmin}_{x\neq y} S_D(x,y)$*. Dann sind a und b Nachbarn in T.*

Zum Beweis sammeln wir weitere Eigenschaften von S_D; zuerst ein Darstellungslemma.

7.10 Lemma (Darstellungslemma)**.** Sei T ein ungewurzelter Binärbaum mit Blättern $L(T)$, Kanten $E(T)$ und gegebenen Kantenlängen $(\ell(e))_{e\in E(T)}$. Sei $D = D_T$ die zugehörige additive Metrik mit Objektmenge $N(D) = L(T)$, und sei S_D wie in (7.1) definiert. Dann lässt sich jedes $S_D(a,b)$ als Linearkombination der Kantenlängen schreiben:

$$S_D(a,b) = \sum_{e\in E(T)} w_e(a,b)\cdot \ell(e)$$

$$\text{mit } w_e(a,b) = \begin{cases} -2 & \text{wenn } e \in P_T(a,b), \\ -2\cdot |L(T)\setminus L_T(a,e)| & \text{sonst,} \end{cases}$$

wobei $P_T(a,b)$ den Pfad zwischen a und b in T bezeichnet, und $L_T(a,e) \subset L(T)$ die Menge der Blätter, die nach Entfernen der Kante e in derselben Zusammenhangskomponente wie a liegen (inklusive a selbst).

Beweis. Da $D(a,b) = \sum_{e\in P_T(a,b)} \ell(e)$ eine Linearkombination der Kantenlängen ist und $S_D(a,b)$ linear in D, ist auch $S_D(a,b)$ eine Linearkombination der $\ell(e)$. Wir berechnen die Gewichte.

Sei $e \in P_T(a,b)$. Nach der Definition von S_D gilt $w_e(a,b) = (n-2) - |\{x : e \in P_T(a,x)\}| - |\{x : e \in P_T(b,x)\}|$. Man gelangt von a nach x über e genau dann wenn man von b nach x die Kante e nicht benutzen muss. Daher gilt $|\{x : e \in P_T(a,x)\}| + |\{x : e \in P_T(b,x)\}| = n$ und man erhält $w_e(a,b) = -2$.

Nun sei $e \notin P_T(a,b)$. Nach der Definition von S_D gilt $w_e(a,b) = -|\{x : e \in P_T(a,x)\}| - |\{x : e \in P_T(b,x)\}|$. Nach Entfernen von e liegen a und b in derselben Zusammenhangskomponente mit der Blattmenge $L_T(a,e) = L_T(b,e)$. Daher muss man zum Erreichen von genau $|L(T)\setminus L_T(a,e)|$ Blättern die Kante e überqueren. □

Beweis. [Satz 7.9] Wenn a überhaupt einen Nachbarn b hat, ist wegen Lemma 7.8 bereits $S_D(a,b) \le S_D(a,x)$ für alle $x \neq a,b$. Der Satz ist also bewiesen, wenn wir zu jedem Paar $\{x,y\}$, wobei sowohl x als auch y *keinen* Nachbarn hat, ein Nachbarpaar $\{a,b\}$ finden mit $S_D(a,b) < S_D(x,y)$.

Wir betrachten Abb. 7.1: Seien x', y' die zu x, y adjazenten inneren Knoten. Sei e die zu x' inzidente Kante auf $P_T(x',y')$. Wenn wir den Pfad $P_T(x',y')$ entfernen, erhalten wir zwei Teilbäume von T, die wir mit T^x und T^y bezeichnen. ObdA sei T^x der kleinere (mit weniger oder gleich vielen Blätter wie T^y); also $|L(T^x)| \le n/2$.

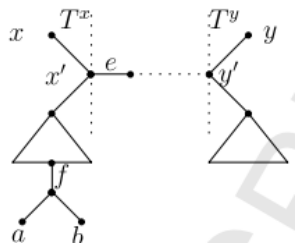


Abbildung 7.1: Skizze zum Beweis von Satz 7.9 (noch zu überarbeiten!)

Da weder x noch y einen Nachbarn hat, muss es sowohl in T^x als auch T^y Nachbarpaare geben. Sei $\{a, b\}$ ein beliebiges Nachbarpaar in T^x; wir zeigen, dass dieses die Bedingung erfüllt. Sei f die vom gemeinsamen Vorfahren wegführende Kante; wir berechnen nun Kantengewichte und behaupten: Es ist $w_g(x, y) \geq w_g(a, b)$ für alle Kanten g. Damit ist nach dem Darstellungslemma dann insbesondere $S_D(x, y) \geq S_D(a, b)$.

Wir unterscheiden zum Beweis die folgenden Fälle (Abbildung 7.1):

- Für alle Blattkanten gilt stets ein Gewicht von -2, also Gleichheit.
- Nichtblattkanten g in T^x (z.B. f) liegen weder auf $P_T(a, b)$ noch auf $P_T(x, y)$, also ist

$$w_g(x, y) = -2(n - |L_T(x, g)|),$$
$$w_g(a, b) = -2(n - |L_T(a, g)|).$$

  Da g innerhalb des kleineren Baums T^x liegt, ist $|L_T(a, g)| \leq |L_T(x, g)|$, un damit auch $w_g(a, b) \leq w_g(x, y)$.
- Kanten g zwischen x' und y' (z.B. e) liegen auf $P_T(x, y)$; daher gilt $w_g(x, y) = -2 \geq w_g(a, b)$.
- Für Nichtblattkanten g in T^y liegen alle vier Knoten a, b, x, y auf derselben Seite von g; daher gilt $w_g(x, y) = w_g(a, b)$. Dasselbe gilt für Kanten, die weder in T^x noch in T^y noch auf dem Pfad zwischen x' und y' liegen.

Wir können die Abschätzung sogar präzisieren: Es ist insbesondere für die Kanten e und f:

- $w_e(x, y) = -2$ und $w_e(a, b) = -2|L(T) \setminus L(T^x)| \leq -n$;
- $w_f(x, y) = -4$ und $w_f(a, b) = -2(n - 2)$.

Setzen wir $\mu(T) := \min_{e \in E(T)} \ell(e)$ als die minimale Kantenlänge in T, dann gilt

$$S_D(x, y) - S_D(a, b) \geq (-2 + n - 4 + 2(n - 2))\mu(T) = (3n - 10)\mu(T) \geq 8\mu(T),$$

da $n \geq 6$, wenn weder x noch y Nachbarn haben. Die zeigt, dass es zu jedem Blattpaar $\{x, y\}$, bei dem sowohl x als auch y keinen Nachbarn hat, stets ein Nachbarpaar $\{a, b\}$ im kleineren Teilbaum T^x gibt, dessen S_D-Wert um mindestens $(3n - 10)\mu(T)$ geringer ist als der von $\{x, y\}$. □

Zusammen mit dem Verschiebungslemma ergibt sich aus dem eben bewiesenen Satz folgende Eigenschaft von S_D: Das Auswahlkriterium ist invariant gegenüber dem Verlängern und Verkürzen einer Blattkante. Ist beispielsweise $\{a, b\}$ das minimale Paar bezüglich D und erhält man $\tilde{D}$ durch eine Veschiedung von a um σ (was einer Verlängerung der Blattkante zu a um σ entspricht), dann ist auch $\{a, b\}$ das minimale Paar bezüglich $\tilde{D}$; denn $S_{\tilde{D}}$ und S_D unterscheiden sich wegen dem Verschiebungslemma um die Konstante 2σ. In beiden Fällen ist $\{\, a, b \,\}$ ein Nachbarpaar.

7.3.5 Berechnung der Kantenlängen und der neuen Distanzen

Wir haben gesehen, dass bei Eingabe einer additiven Matrix im ersten NJ-Schritt tatsächlich ein Nachbarpaar ausgewählt wird. Um weitere Aussagen über NJ zu treffen, müssen wir noch spezifizieren, wie genau die nächste Metrik $D' \equiv D_{i+1}$ aus $D \equiv D_i$ berechnet wird (vgl. Algorithmus 2).

Wir fassen das in Schritt i neu gebildete Blatt c_i als Wurzel oder Vorfahren von $\{a_i, b_i\}$ auf und berechnen mit Hilfe einer Skizze

$$D'(c_i, x) := [D(a_i, x) + D(b_i, x) - D(a_i, b_i)]/2. \tag{7.2}$$

Die Kantenlängen $\ell(a_i, c_i)$ und $\ell(b_i, c_i)$ ergeben sich dann als

$$\ell(a_i, c_i) = D(a_i, x) - D'(c_i, x) = [D(a_i, x) - D(b_i, x) + D(a_i, b_i)]/2 \ (x \neq a_i, b_i, c_i \text{ beliebig}).$$

Bei additiven Distanzen D ergibt sich für jede Wahl von x der gleiche Wert. (Hieran kann man Additivität testen!) Allgemein (bei nur fast oder nicht additiven Distanzen) ist es sinnvoll, den Mittelwert über alle $x \neq a_i, b_i$ zu bilden; es ergibt sich

$$\ell(a_i, c_i) := [R(a_i) - R(b_i)]/2(n-2) + D(a_i, b_i)/2, \tag{7.3}$$
$$\ell(b_i, c_i) := [R(b_i) - R(a_i)]/2(n-2) + D(a_i, b_i)/2. \tag{7.4}$$

7.11 Lemma. Sei D additiv; dazu gehöre Baum T. Sei $\{\, a, b \,\}$ das Nachbarpaar in T, das von NJ zusammengefasst wird und c der Vorfahre. Die gemäß (7.2) berechneten Distanzen $D'(c, x)$ die Summen der Pfadlängen zwischen dem inneren Knoten c und dem Blatt x in T. Dann sind $\ell(a, c)$ und $\ell(b, c)$ gemäß (7.3) und (7.4) die Kantenlängen von (a, c) bzw. (b, c) in T. Es ist also

$$\begin{aligned} D(a, b) &= \ell(a, c) + \ell(b, c) \\ D(a, x) &= \ell(a, c) + D'(c, x) \text{ für alle } x \neq a, b \\ D(b, x) &= \ell(b, c) + D'(c, x) \text{ für alle } x \neq a, b. \end{aligned}$$

Beweis. Dass NJ überhaupt ein Nachbarpaar zusammenfasst, folgt aus Satz 7.9. Dass $D'(c, x)$ wie berechnet mit der Summe der Pfadlänge zwischen c und x übereinstimmt, wurde oben bereits gezeigt. Dass die berechneten $\ell(a, c)$ und $\ell(b, c)$ den Kantenlängen in T entsprechen, wurde ebenfalls bereits nachgerechnet: für alle x gilt $\ell(a, c) = D(a, x) - D'(c, x)$. Daraus folgen dann auch die verbleibenden Formeln. □

Als einfache Folgerung erhalten wir die Tatsache, dass $D' = D_{i+1}$ wieder eine additive Distanzmatrix ist, und zwar zum gleichen Baum wie D_i, wobei nur die Blätter a_i und b_i durch ihren Vorfahren ersetzt wurden.

7.12 Lemma. Sei D eine additive Metrik und D' die nach einem NJ-Schritt reduzierte Distanzmatrix. Dann ist D' additiv.

Beweis. Sei T der Baum zu D, und sei $\{a_i, b_i\}$ das im NJ-Schritt ausgewählte Blattpaar. Sei c_i in T der Vorfahre von a_i, b_i und sei c_i' der verbleibenden zu c_i adjazente Knoten. Wir betrachten den Baum $S = T \setminus \{a_i, b_i\}$, in dem c_i nun ein Blatt ist, bei dem sich die Längen der verbleibenden Kanten gegenüber T nicht ändern. Wir behaupten D' enthält die paarweisen Blattabstände in S und ist damit additiv. Dies haben wir aber mit Lemma 7.11 schon gezeigt. □

7.3.6 Korrektheit des NJ-Algorithmus

Der folgende Satz fasst zusammen, dass NJ auf additiven Distanzen den korrekten Baum berechnet.

7.13 Satz. *Sei T ein ungewurzelter Binärbaum mit gegebenen Kantenlängen $\ell_T(e)$ und D die zugehörige additive Metrik. Dann rekonstruiert NJ(D) den Baum T mit den korrekten Kantenlängen.*

Beweis. Satz 7.9 liefert, dass NJ ein Nachbarpaar auswählt. Die Wahl der Kantenlängen $\ell(a_i, c_i)$ und $\ell(b_i, c_i)$ in Abschnitt 7.3.5 stellt sicher, dass die Kantenlängen bei einer additiven Metrik korrekt gewählt werden. Lemma 7.12 liefert, dass die reduzierte Metrik wieder additiv ist. □

7.4 Neighbor Joining auf fast additiven Distanzen

Der "Spielraum" von $(3n - 10)\mu(T)$ im Beweis von Satz 7.9 liefert einen Hinweis, dass NJ noch besser ist als bisher gezeigt, und auch auf nicht perfekt additiven Distanzen das korrekte Ergebnis liefert.

7.14 Definition. Eine Metrik D heißt *fast additiv*, wenn es einen Baum T mit induzierter additiver Metrik D_T gibt, so dass

$$\|D - D_T\|_\infty := \max_{x,y} |D(x, y) - D_T(x, y)| < \mu(T)/2,$$

wobei wieder $\mu(T) := \min_{e \in E(T)} \ell(e)$. Man sagt dann auch (aufgrund der Eindeutigkeit von T; siehe folgendes Lemma), D ist fast additiv in Bezug zu T, oder D ist eine zu T gehörende fast additive Metrik.

7.15 Lemma (Eindeutigkeit des Bezugsbaums)**.** Sei D fast additiv in Bezug zu S und zu T. Dann stimmen die Topologien von S und T überein.

Beweis. Seien D_T und D_S die zu T und S gehördenden additiven Metriken. Sei $(ij : kl)$ ein Quartett in T. Wir zeigen, dass $(ij : kl)$ auch ein Quartett in S ist. Dies gilt für alle Quartette, und auch mit vertauschten Rollen von S und T. Damit stimmen die Mengen der Quartette von S und T überein, und damit auch die Baumtopologien.

Unter der Voraussetzung, dass $(ij : kl)$ ein Quartett in T ist, zeigen wir $D_S(i,j)+D_S(k,l) < D_S(i,k) + D_S(j,l) + 2\mu(S)$:

$$\begin{aligned}
D_S(i,j) + D_S(k,l) &< D(i,j) + D(k,l) + \mu(S) && (D \text{ fast additiv zu } S)\\
&< D_T(i,j) + D_T(k,l) + \mu(S) + \mu(T) && (D \text{ fast additiv zu } T)\\
&\leq D_T(i,k) + D_T(j,l) + \mu(S) - \mu(T) && (\text{4-Punkte-Bedingung in } T)\\
&< D(i,k) + D(j,l) + \mu(S) && (D \text{ fast additiv zu } T)\\
&< D_S(i,k) + D_S(j,l) + 2\mu(S) && (D \text{ fast additiv zu } S)
\end{aligned}$$

Angenommen, es sei $(ik : jl)$ ein Quartett in S; dann würde folgen, dass

$$\begin{aligned}
D_S(i,k) + D_S(j,l) &\leq D_S(i,j) + D_S(k,l) - 2\mu(S) && (\text{4-Punkte-Bedingung in } S)\\
&< D_S(i,k) + D_T(j,l) && (\text{obige Rechnung});
\end{aligned}$$

ein Widerspruch. Daher ist $(ik : jl)$ kein Quartett in S. Eine analoge Rechnung zeigt, dass auch $(il : jk)$ kein Quartett in S ist. Daher muss $(ij : kl)$ ein Quartett in S sein, was zu zeigen war. □

Die zu beweisende Verallgemeinerung von Satz 7.9 lautet nun

7.16 Satz (Atteson, 1999; Elias & Lagergren, 2008)**.** *Sei T ein ungewurzelter Binärbaum mit gegebenen Kantenlängen, D eine zu T gehörende fast additive Metrik, und S_D wie in* (7.1) *definiert. Sei $\{a,b\} := \operatorname{argmin}_{x\neq y} S_D(x,y)$. Dann sind a und b Nachbarn in T.*

Zum Beweis müssen wir einige technische Vorbereitungen treffen. Das nächste Lemma zeigt, dass sich S_D und S_{D_T} nicht beliebig unterscheiden können, wenn man die Differenz auf drei oder vier Punkten betachtet.

7.17 Lemma (Technisches Lemma)**.** Sei T ein ungewurzelter Binärbaum mit gegebenen Kantenlängen, D_T die von T induzierte additive Metrik, und D eine zu T gehörende fast additive Metrik. Sei $\mu(T) := \min_e \ell(e)$, und seien a, b, x, y Objekte aus $N(D)$. Dann gilt:

$$\begin{aligned}
&S_D(a,b) - S_{D_T}(a,b) + S_{D_T}(x,y) - S_D(x,y)\\
&> \begin{cases} -3(n-4)\cdot\mu(T), & \text{wenn } a,b,x,y \text{ paarweise verschieden,}\\ -2(n-3)\cdot\mu(T), & \text{wenn } |\{a,b\}\cap\{x,y\}| = 1. \end{cases}
\end{aligned}$$

Beweis. Der Fall, dass a, b, x, y verschieden sind, ist elementar mit Hilfe der Definition von S_D nachzurechnen. Es ist für $a \neq b$:

$$\begin{aligned}
|S_D(a,b) - S_{D_T}(a,b)| &= |[(n-2)D(a,b) - R(a) - R(b)] - [(n-2)D_T(a,b) - R_T(a) - R_T(b)]|\\
&\leq (n-2)|D(a,b) - D_T(a,b)| + |R(a) - R_T(a)| + |R(b) - R_T(b)|\\
&< (n-2)\mu(T)/2 + (n-1)\mu(T)/2 + (n-1)\mu(T)/2\\
&= 3n - 4\mu(T)/2.
\end{aligned}$$

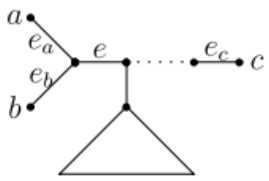


Abbildung 7.2: Skizze zum Beweis von Lemma 7.19 (noch zu überarbeiten!)

Dieselbe Abschätzung ergibt sich für x, y, also insgesamt $> -(3n-4)\mu(T) > -3(n-4)\mu(T)$, wie behauptet.

Im anderen Fall sei oBdA $b = x$, so dass wir einen Sternbaum mit den 3 Blättern $a, b = x, y$ betrachten; der innere Knoten sei u. Nun ist

$$\begin{aligned} S_D(a,b) - S_{D_T}(a,b) + S_{D_T}(b,y) - S_D(b,y) &= [(n-2)D(a,b) - R(a) - R(b)] - [(n-2)D_T(a,b) - R_T(a) - R_T \\ &= (n-2)[D(a,b) - D_T(a,b) + D_T(y,b) - D(y,b)] + [R(a) - R_T(a) \\ &> (n-2)(-2\mu(T)/2) + (n-1)(-2\mu(T)/2) \\ &= -(2n-3)\mu(T) \\ &> -2(n-3)\mu(T), \end{aligned}$$

wie behauptet. □

Das folgende Sichtbarkeitslemma verallgemeinert Lemma 7.8 auf fast additive Distanzen. Zunächst definieren wir:

7.18 Definition (Sichtbarkeit)**.** Der Knoten b oder das Paar $\{a, b\}$ heißt *sichtbar von a aus*, wenn $S_D(a,b) = \min_{x \neq a} S_D(a,x)$. Das Paar $\{a,b\}$ heißt *sichtbar*, wenn es von a oder b sichtbar ist.

7.19 Lemma (Sichtbarkeitslemma)**.** Sei T ein ungewurzelter Binärbaum auf n Taxa mit gegebenen Kantenlängen $\ell(e)$ mit $\mu(T) := \min_e \ell(e) > 0$. Sei D eine zu T gehörende fast additive Metrik, und S_D wie in (7.1) definiert. Wenn a, b Nachbarn in T sind, dann ist $\{a,b\}$ von a aus sichtbar.

Beweis. Wir zeigen $S_D(a,c) > S_D(a,b)$ für alle $c \neq a, b$. Dazu sei D_T die von T induzierte additive Metrik. Wir zerlegen

$$\begin{aligned} &S_D(a,c) - S_D(a,b) \\ &= [S_D(a,c) - S_{D_T}(a,c) + S_{D_T}(a,b) - S_D(a,b)] + [S_{D_T}(a,c) - S_{D_T}(a,b)] \end{aligned}$$

und sehen, dass die erste Klammer nach dem technischen Lemma $> -2(n-3)\mu(T)$ ist.

Nach Lemma 7.8 ist auch die zweite Klammer zumindest nicht negativ. Wir zeigen jetzt, dass sie sogar $\geq 2(n-3)\mu(T)$ ist; damit gilt dann $S_D(a,c) - S_D(a,b) > 0$, was zu zeigen war.

Wir wählen also ein $c \neq a, b$ und benennen die Blattkanten zu a, b, c jeweils e_a, e_b, e_c. Da a und b Nachbarn sind, gibt es genau eine dritte Kante e, die zum gemeinsamen Vorfahren von a und b inzident ist und nicht zu a oder b führt (Abbildung 7.2).

Wir berechnen jetzt mit Hilfe des Darstellungslemmas wieder einige Kantengewichte:

- Für Blattkanten $f \in \{e_a, e_b, e_c\}$ gilt stets $w_f(a,c) = w_f(a,b) = -2$.
- Für Kanten $f \in P_T(a,c)$, $f \neq e_a, e_c$ gilt $w_f(a,c) = -2 > -4 \geq= w_f(a,b)$. Insbesondere gilt $w_e(a,c) = -2 > -2(n-2) = w_e(a,b)$.
- Für Kanten $f \notin P_T(a,c)$ gilt stets $w_f(a,c) = w_f(a,b)$.

Also sind für alle f die Gewichtsdifferenzen $w_f(a,c) - w_f(a,b) > 0$ und daher

$$\begin{aligned} & S_D(a,c) - S_D(a,b) \\ = & \sum_f [w_f(a,c) - w_f(a,b)] \cdot \ell(f) \\ \geq & \mu(T) \cdot \sum_f [w_f(a,c) - w_f(a,b)] \\ \geq & \mu(T) \cdot (-2 + 2(n-2)) = 2(n-3)\mu(T). \end{aligned}$$

□

Mit Hilfe dieses Lemmas können wir den Beweis von Satz 7.16 nun führen.

Beweis. [Satz 7.16] Wir gehen vor wie beim Beweis von Satz 7.9 in Abschnitt 7.3.4: Wenn a überhaupt einen Nachbarn hat, ist wegen dem Sichtbarkeitslemma (Lemma 7.19) bereits $S_D(a,b) < S_D(a,x)$ für alle $x \neq a, b$. Der Satz ist also bewiesen, wenn wir zu jedem Paar $\{x,y\}$, wobei sowohl x als auch y keine Nachbarn hat, ein Nachbarpaar $\{a,b\}$ finden mit $S_D(x,y) > S_D(a,b)$.

Sei wieder D_T die von T induzierte additive Metrik. Wir zerlegen

$$\begin{aligned} & S_D(x,y) - S_D(a,b) \\ = & [S_D(x,y) - S_{D_T}(x,y) + S_{D_T}(a,b) - S_D(a,b)] + [S_{D_T}(x,y) - S_{D_T}(a,b)] \\ > & -3(n-4)\mu(T) + (3n-10)\mu(T) \\ = & 2\mu(T) > 0, \end{aligned}$$

wobei wir das technische Lemma 7.17 und den Beweis von Satz 7.9 auf S_{D_T} angewendet haben. □

Um die Korrektheit von NJ auf fast additiven Distanzen zu beweisen, benötigen wir noch, dass nach einem NJ-Schritt die neue Distanzmatriz ebenfalls wieder fast additiv ist.

7.20 Lemma. Sei D eine fast additive Metrik und D' die nach einem NJ-Schritt reduzierte Distanzmatrix. Dann ist D' fast additiv.

Beweis. Dies zeigen wir wie im Beweis von Lemma 7.12 durch denselben Baum S wie dort; wir müssen nachweisen, dass $|D'(x,y) - D_S(x,y)| < \mu(S)/2$ für alle x, y unter der Voraussetzung, dass $|D(x,y) - D_T(x,y)| < \mu(T)/2$ für alle x, y.

Nun ist $\mu(S) \geq \mu(T)$ und somit für $x, y \neq c_i$ bereits $D'(x,y) = D(x,y)$ und $D_S(x,y) = D_T(x,y)$, also $|D'(x,y) - D_S(x,y)| = |D(x,y) - D_T(x,y)| < \mu(T)/2 \leq \mu(S)/2$.

Es bleiben die Distanzen zu $c = c_i$ zu betrachten. Sei c' der einzige zu c adjazente Knoten in S. Dann ist

$$\begin{aligned}
&|D'(x,c) - D_S(x,c)| \\
&= |(D(x,a) + D(x,b))/2 - D_S(c',x) - \ell_S(c,c')| \\
&= |(D(x,a) + D(x,b))/2 - D_T(c',x) - \ell_T(c,c') - (\ell_T(c,a) + \ell_T(c,b))/2| \\
&= |(D(x,a) - D_T(x,a))/2 + (D(x,b) - D_T(x,b))/2| \\
&< \mu(T)/4 + \mu(T)/4 \leq \mu(S)/2,
\end{aligned}$$

was zu zeigen war. □

7.21 Satz. *Sei T ein ungewurzelter Binärbaum mit gegebenen Kantenlängen $\ell_T(e)$ und eine D zugehörige fast additive Metrik. Dann rekonstruiert NJ(D) die Baumtopologie von T.*

Beweis. Die Korrektheit folgt aus dem Satz von Atteson (Satz 7.16) und aus Lemma 7.20. □

Zusammenfassung der Beweisstrategie. Wir fassen noch einmal die Beweisstrategie für die Korrektheit von NJ auf (fast) additiven Metriken D mit zugehörigem Baum T zusammen.

1. Wenn a und b Nachbarn in T sind, ist $S_D(a,b) = \min_{x \neq a} S_D(a,x)$; d.h., b ist von a aus sichtbar [Lemma 7.8 bzw. Sichtbarkeitslemma].
2. Betrachtet man $\{a,b\} = \operatorname{argmin}_{x \neq y} S_D(x,y)$, dann sind a und b in T Nachbarn [Satz 7.9 bzw. Satz 7.16].
3. Fügt man die so bestimmten a, b zu c zusammen und berechnet die sich ergebenden neuen Distanzen D', so ist D' ebenfalls wieder (fast) additiv [Satz 7.13 bzw. Satz 7.21].

7.5 Fast Neighbor Joining

Neighbor Joining ist nach dem bisher Gesagten also ein Algorithmus mit der Eigenschaft, dass er robust gegenüber kleinen Fehlern in additiven Eingabedistanzen ist. Ungünstig ist die Tatsache, dass eine direkte Implementierung von Algorithmus 2 auf n Taxa eine Laufzeit von $O(n^3)$ hat. Wir zeigen nun, wie diese auf $O(n^2)$ reduziert werden kann, so dass die oben genannten Eigenschaften (korrekter Baum auf fast additiven Eingabedistanzen) erhalten bleiben. Das resultierende Verfahren heißt demzufolge *Fast Neighbor Joining* (FNJ). Auf Distanzen, die "weit weg" von additiven Distanzen sind, können NJ und FNJ verschiedene Bäume liefern, und es gibt keinen Konsens darüber, welches Verfahren in diesem Fall "besser" ist. In der Regel wird man sich daher für das schnellere FNJ entscheiden.

Der erste Trick besteht darin, das Minimum von $S_D(x,y)$ nicht über alle Paare $\{x,y\}$ zu suchen, sondern nur über die sichtbaren Paare. Beim Neuberechnen der Distanzen müssen wir dann sicherstellen, dass die Menge ebenfalls korrekt aktualisiert wird. Der zweite Trick besteht in der effizienten Aktualisierung der Summen $R(x)$ in jedem Schritt, so dass insgesamt jede der $n-3$ Iterationen in $O(n)$ Zeit erfolgen kann.

Algorithm 3 FNJ(D): berechnet aus der Metrik D auf der Objektmenge $N(D)$ mit $n := |N(D)| \geq 3$ einen ungewurzelten Baum in $O(n^2)$ Zeit

Initialisiere $R(a) \leftarrow \sum_x D(a,x)$ für alle $a \in N(D)$
Initialisiere sichtbare Menge $V \leftarrow \{\{a, \operatorname{argmin}_{x \neq a} S_D(a,x)\} : a \in N(D)\}$
for $i \leftarrow 1, \ldots, n-3$ **do**
    $\{a,b\} \leftarrow \operatorname{argmin}_{\{x,y\} \in V} [(|N(D)|-2) \cdot D(x,y) - R(x) - R(y)]$
    Reduziere $\{a,b\}$ auf ein neues Blatt c und berechne Kantenlängen
        $\ell(a,c) \leftarrow [R(a) - R(b)]/2(|N(D)|-2) + D(a,b)/2$,
        $\ell(b,c) \leftarrow [R(b) - R(a)]/2(|N(D)|-2) + D(a,b)/2$.
    Aktualisiere D auf neue Blattmenge $N(D) \leftarrow N(D) \setminus \{a,b\} \cup \{c\}$ durch
      $D(c,x) \leftarrow [D(a,x) + D(b,x) - D(a,b)]/2$ für alle $x \neq a, b, c$.
    Berechne $R(c) \leftarrow \sum_x D(c,x)$.
    Für alle $y \neq c$, aktualisiere $R(y) \leftarrow R(y) - D(a,y) - D(b,y) + D(c,y)$
    $c' \leftarrow \operatorname{argmin}_{y \neq c} [(|N(D)|-2) \cdot D(c,y) - R(c) - R(y)]$
    Aktualisiere $V \leftarrow V \setminus \{\{a,x\}, \{b,x\} : x \in N(D)\} \cup \{\{c,c'\}\}$
Verbinde die 3 verbleibenden Objekte zu einem Sternbaum
Berechne die Kantenlängen des Sternbaums
return Sternbaum bzw. dessen Wurzel

Algorithmus 3 zeigt die einzelnen Schritte, die wir im Folgenden analysieren. Zunächst die Laufzeit: Die Initialisierung ist in $O(n^2)$ Zeit problemlos möglich; in der for-Schleife benötigt jeder einzelne Schritt nur konstante oder $O(n)$ Zeit.

Der Unterschied zu NJ besteht nur darin, dass in Zeile 4 über die sichtbare Menge V statt über alle Paare $x \neq y$ minimiert wird und $S_D(x,y)$ on-the-fly aus D und R in konstanter Zeit berechnet wird. In den Zeilen ... sieht man, wie R für jedes verbleibende Blatt in konstanter Zeit aktualisiert wird.

Wir wollen nun zeigen, dass FNJ auf (fast) additiven Distanzen das gleiche Ergebnis liefert wie NJ (nämlich den korrekten Baum). Dazu müssen wir nur sicherstellen, dass nach jeder Iteration das nächste Nachbarpaar, das NJ im nächsten Schritt auswählen würde, in V ist.

7.22 Satz. *Auf (fast) additiven Distanzen liefert FNJ denselben Baum wie NJ.*

Beweis. Das Sichtbarkeitslemma für fast additive Distanzen garantiert zu Beginn der ersten Iteration, dass *alle* Blatt-Nachbarpaare des korrekten Baums T (der von NJ rekonstruiert wird), in V liegen. Satz 7.16 garantiert, dass das ausgewählte Paar ein Nachbarpaar ist. Durch die Initialisierung von V ist klar, dass FNJ und NJ im ersten Schritt dasselbe Nachbarpaar (a,b) auswählen und reduzieren.

Nach einem Reduktionsschritt ist die resultierende Distanz wieder fast additiv. Es gibt dann zwei Möglichkeiten: (1) Das nächste von NJ ausgewählte Paar enthält den neuen Knoten c; dann muss es sich wieder nach dem Sichtbarkeitslemma um das gerade zu V hinzugefügte $\{c,c'\}$ handeln; (2) das nächste Paar enthält c nicht; dann muss es aber nach dem Sichtbarkeitslemma schon vorher in V gewesen sein – aus V werden keine Paare entfernt, es sei denn, einer ihrer Knoten wurde reduziert.

Durch Induktion folgt, dass in jeder Iteration das Paar, das von NJ ausgewählt wird, bei FNJ in V liegt und somit auch von FNJ ausgewählt wird. □

7.6 Weitere Resultate zum Neighbor-Joining-Algorithmus*

Über NJ und FNJ lässt sich noch mehr beweisen: Mihaescu, Levy und Pachter haben untersucht, was passiert, wenn die Eingabemetrik nicht mehr fast additiv ist und führen die Begriffe "Quartett-additiv" und "Quartett-konsistent" ein.

7.23 Definition. Eine Distanzmatrix D heißt *Quartett-konsistent* mit einem Baum T, wenn für alle Nachbarquartette $(ij : kl) \in T$ gilt, dass $w_D(ij : kl) > \max\{w_D(ik : jl), w_D(il : jk)\}$ (siehe Definition 7.5).

7.24 Definition. Ein Blatt x liegt *innerhalb* eines Nachbarquartetts $(ij : kl)$, wenn weder $(ik : xl)$ noch $(ik : xj)$ Nachbarquartette sind.

7.25 Definition. Eine Distanzmatrix D heißt *Quartett-additiv* in Bezug auf einen Baum T, wenn für jedes Nachbarquartett $(ij : kl) \in T$ und jedes x innerhalb von $(ij : kl)$ und jedes y nicht innerhalb von $(ij : kl)$, so dass $(ij : xy) \notin T$, gilt: $w_D(kl : xy) > w_D(ij : xy)$.

Das entscheidende Resultat lautet wie folgt.

7.26 Satz. *Sei D eine Quartett-additive und Quartett-konsistente Distanzfunktion bezüglich eines Baums T. Dann wird von NJ und FNJ der Baum T ausgegeben.*

Dieses Resultat stellt einen Bezug zwischen einem Baum und den enthaltenen Nachbarquartetten und zeigt, dass (F)NJ noch robuster als bisher bekannt gegenüber nicht-additiven Distanzen ist.

Ein weiteres interessantes Resultat erhält man, indem man einzelne Kanten betrachtet.

7.27 Definition. Sei T ein ungewurzelter Baum mit Blattmenge $L(T)$. Sei e eine innere Kante. Löschen von e induziert eine Bipartition $L(T) = P \cup Q$ mit $P \cap Q = \{\}$, $|P| \geq 2, |Q| \geq 2$, so dass P bzw. Q die Blätter in den beiden Zusammenhangskomponenten sind. Wir nennen diese Bipartition den zu e gehörenden *Split* $P|e|Q$.

Sei weiter D_T die von T induzierte additive Metrik, und sei ℓ die Länge von e. Eine Distanzmatrix D heißt *$P|e|Q$-additiv bezüglich T*, wenn

- $D(x, y) - D_T(x, y) < \ell/4$ für alle $(x, y) \in P \times P$ und alle $(x, y) \in Q \times Q$,
- $|D(x, y) - D_T(x, y)| < \ell/4$ für alle $(x, y) \in P \times Q$.

7.28 Satz. *Sei T ein Baum mit einem Split $P|e|Q$. Sei D eine $P|e|Q$-additive Distanz bezüglich T. Dann rekonstruieren NJ und FNJ jeweils einen Baum, der ebenfalls den Split $P|e|Q$ enthält.*

Mit anderen Worten, solange die Unterschiede der Distanzen D und D_T entlang e nicht zu stark sind und innerhalb von P und Q die Distanzen in D nicht zu viel größer sind als die in D_T, wird zumindest die Existenz der Kante e von NJ und FNJ korrekt rekonstruiert.

Literatur Dieser Abschnitt muss noch mit BibTeX aufgearbeitet werden.

- Saitou N, Nei M (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol* **4** (4): 406–425. PMID 3447015. `http://mbe.oxfordjournals.org/cgi/reprint/4/4/406`.
- Studier JA, Keppler KJ (1988). A note on the Neighbor-Joining algorithm of Saitou and Nei. *Mol Biol Evol* **5** (6): 729–731. PMID 3221794. `http://mbe.oxfordjournals.org/cgi/reprint/5/6/729.pdf`.
- Mihaescu R, Levy D, Pachter L (2006). Why neighbor-joining works.
- Elias I, Lagergren J (2008). Fast neighbor joining. *Theoretical Computer Science*

Literaturverzeichnis

A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

V. Boeva, J. Clément, M. Régnier, M. A. Roytberg, and V. J. Makeev. Exact p-value calculation for heterotypic clusters of regulatory motifs and its application in computational annotation of cis-regulatory modules. *Algorithms for Molecular Biology*, 2:13, October 2007. ISSN 1748-7188.

P. Brémaud. *Markov chains, Gibbs fields, Monte Carlo simulation, and queues.* Springer, 1999.

S. Dori and G. M. Landau. Construction of Aho Corasick automaton in linear time for integer alphabets. *Information Processing Letters*, 98(2):66–72, 2006.

J. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi and A. Paz, editors, *The theory of machines and computations*, pages 189–196. Academic Press, New York, 1971.

N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. S. Langendijk-Genevaux, M. Pagni, and C. J. A. Sigrist. The PROSITE database. *Nucleic Acids Research*, 34(S1): D227–230, 2006.

T. Knuutila. Re-describing an algorithm by Hopcroft. *Theoretical Computer Science*, 250: 333–363, 2001.

G. Kucherov, L. Noé, and M. Roytberg. A unifying framework for seed sensitivity and its application to subset seeds. *J. Bioinform. Comput. Biol.*, 4(2):553–569, 2006.

M. Lladser, M. D. Betterton, and R. Knight. Multiple pattern matching: A Markov chain approach. *Journal of Mathematical Biology*, 56(1-2):51–92, 2008.

M. Lothaire. *Applied Combinatorics on Words (Encyclopedia of Mathematics and its Applications).* Cambridge University Press, 2005.

T. Marschall. Construction of minimal deterministic finite automata from biological motifs. *Theoretical Computer Science*, 412:922–930, 2011.

G. Navarro and M. Raffinot. *Flexible pattern matching in strings*. Cambridge University Press, 2002.

P. Nicodème, B. Salvy, and P. Flajolet. Motif statistics. *Theoretical Computer Science*, 287: 593–617, 2002.

G. Nuel. Pattern Markov chains: optimal Markov chain embedding through deterministic finite automata. *Journal of Applied Probability*, 45:226–243, 2008.

M. Régnier. A unifed approach to word occurrence probabilities. *Discrete Applied Mathematics*, 104:259–280, 2000.

G. Reinert, S. Schbath, and M. S. Waterman. Probabilistic and statistical properties of words: An overview. *Journal of Computational Biology*, 7(1-2):1–46, 2000.

M. Schulz, D. Weese, T. Rausch, A. Döring, K. Reinert, and M. Vingron. Fast and adaptive variable order markov chain construction. In K. A. Crandall and J. Lagergren, editors, *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics (WABI), Karlsruhe, Germany*, volume 5251 of *LNCS*, pages 306–317, 2008. URL `http://dx.doi.org/10.1007/978-3-540-87361-7_26`.

J. Zhang, B. Jiang, M. Li, J. Tromp, X. Zhang, and M. Q. Zhang. Computing exact p-values for DNA motifs. *Bioinformatics*, 23(5):531–537, 2007.