Grundlegende Algorithmen der Bioinformatik

Vortragsausarbeitungen des gleichnamigen Proseminars

LS 11, Fakultät für Informatik, TU Dortmund

Prof. Dr. Sven Rahmann

mit Beiträgen der TeilnehmerInnen

Wintersemester 2008/09

Entwurf, 5. Mai 2009

Inhaltsverzeichnis

Vorbemerkungen

Dieses Dokument enthält die Vortragsausarbeitungen des Proseminars Grundlegende Algorithmen der Bioinformatik, das ich im Wintersemester 2008/09 an der Fakultät für Informatik an der TU Dortmund angeobten habe.

Ziel war es, anhand von Bioinformatik-Lehrbüchern grundlegende Algorithmen der Bioinformatik zu erarbeiten, vorzustellen und kurz zusammenzufassen, so dass die wesentlichen Ideen hinter den vorgestellten Verfahren deutlich werden.

Als Veranstalter hoffe ich, dass alle Teilnehmer viel gelernt haben und Spaß bei der Arbeit hatten.

– Prof. Dr. Sven Rahmann, TU Dortmund

KAPITEL 2

Dotplots: Darstellung und Filterung

Ausarbeitung von Sven Radetzky, WiSe 2008/09

2.1 Einleitung

Dieses Kapitel dient als Einführung für Verfahren zum Sequenzvergleich durch das Thema Dotplots. Anders als die nachfolgenden Kapitel dienen Dotplots und die damit verwandten Verfahren nicht der Lösung eines klaren algorithmischen Problems, und obwohl hier zwei Algorithmen vorgestellt werden, sind sie nur bezeichnet als Algoritmen aus Tradition anstatt aus akurateren Gründen; viel passender wäre die Bezeichnung Verfahren oder Heuristiken. Dotplots sind ein mittlerweile über 30 Jahre altes Werkzeug, um sich schnell und anschaulich einen Überblick über Gemeinsamkeiten zweier oder mehrerer Sequenzen zu verschaffen. Die Auswertung von Dotplots bedarf allerdings entweder eines Menschen oder weiterer Verfahren oder Algorithmen, auf die wir hier nicht weiter eingehen werden.

2.2 Dotplots

Der klassische Dotplot basiert auf einer Arbeit von Gibbs und McIntyre aus dem Jahre 1970 [Gibbs (1970)]. Dotplots sind Matrizen die im klassischsten Sinne für den Vergleich von zwei beliebigen Strings genutzt werden. In der Bioinformatik vor allem für den Vergleich von DNA- als auch Proteinsequenzen, welche Strings über dem Alphabet $\Sigma = \{A, C, G, T\}$ für DNA oder
$\Sigma = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$ für Proteinsequenzen.

Das klassische Verfahren ist recht simpel, eine Sequenz steht für die Spalten die andere für die Zeilen und an jeder Position i, j in der von den Sequenzen aufgespannten Matrix, wo die Sequenzen gleich sind, wird ein Punkt gesetzt, bzw. eine Eins oder falls nicht, kein

	S	*E*	*Q*	*U*	*E*	*N*	*C*	*E*	*A*	*N*	*A*	*L*	*Y*	*S*	*I*	*S*	*P*	*R*	*I*	*M*	*E*	*R*
S	1													1		1						
E		1			1			1													1	
Q			1																			
U				1																		
E		1			1			1													1	
N						1				1												
C							1															
E		1			1			1													1	
A									1		1											
N						1				1												
A									1		1											
L												1										
Y													1									
S	1													1		1						
I															1				1			
S	1													1		1						
P																	1					
R																		1				1
I															1				1			
M																				1		
E		1			1			1													1	
R																		1				1

Tabelle 2.1: einfacher Dotplot zur Veranschaulichung (aus Gribskov und Devereux, *Sequence Analysis Primer*, 1991) die leeren Zellen entsprechen den Nullen.

Punkt, bzw. eine Null wie in Tabelle 2.1. Vor allem wegen der Länge der meisten Sequenzen, welche in der Bioinformatik Tausende oder auch Zehntausende von Zeichen umfassen können, werden Dotplots meistens als Pixelmatrizen dargestellt, welche oft auf eine Größe geschrumpft werden, welche es einem Betrachter erlaubt, das gesamte Dotplot mit einem Blick zu betrachten. Das Verfahren ist so oder so nicht gut dafür geeignet, genaue Analysen auf großen Sequenzen zu betreiben, weshalb der Detailverlust bei dieser Darstellung meistens keine Probleme bereitet.

Auch wenn das generelle Verfahren dasselbe ist, gibt es doch seit 1970 Veränderungen. Es ist üblich nun Dotplots mit Hilfe von **Substitutionsmatrizen** zu berechnen, um auch biologische Ähnlichkeiten in deren Darstellung zu berücksichtigen. Das heißt man hat eine weitere Matrix zu Hand, welche über dem den Sequenzen zugrunde liegenden Alphabet aufgebaut ist, und für jede mögliche Zeichenkombination bestimmte Werte vorschreibt. Mit Substitutionsmatrizen werden die Punkte in Dotplots nicht mehr nur an identischen Stellen gesetzt sondern auch an den Stellen, welche einen gewissen Mindestwert nach der verwendeten Substitutionsmatrix erhalten. Tabelle 2.2 zeigt eine Matrix für DNA-Sequenzen, während Matrizen für Proteinsequenzen und auch mehr Informationen zu diesem Thema im gleichnamigen Wikipedia Artikel gefunden werden können [Wikipedia (a)].

Formal ist ein einfacher Dotplot zweier Sequenzen X, Y mit $|X| = n$ und $|Y| = m$, über einem Alphabet Σ mit einer Substitutionsmatrix $S = I^{|\Sigma|}$, also der Identitätsmatrix der

	A	C	G	T
A	4	-5	-5	-5
C	-5	4	-5	-5
G	-5	-5	4	-5
T	-5	-5	-5	4

Tabelle 2.2: Gebräuchliche Substitutionsmatrix für DNA Sequenzen

$|\Sigma|$-ten Dimension, definiert als Matrix D mit

$$D_{i,j} := \begin{cases} 1 & \text{falls} x_i = y_j \\ 0 & \text{sonst} \end{cases} \tag{2.1}$$

Nun werden noch kurz einige Sachen zur **Interpretation** von Dotplots erläutert. Linien auf der Hauptdiagonalen eines Dotplots zeigen offensichtlich exakte Übereinstimmungen, sowohl an den Zeichen als auch an den Positionen in den entsprechenden Sequenzen. Allgemein zeigen diagonale Linien identische Teilsequenzen an. Wobei besonders zu erwähnen ist, daß diagonale Linien in Leserichtung allerdings von unten nach oben, das vorhandensein von inversierten Teilsequenzen anzeigen. Es lassen sich auch leicht Sequenzwiederholungen erkennen, welche durch übereinander- oder nebeneinanderliegenden diagonalen Linien dargestellt werden. Weiterhin lassen sich auch einfach Rekombinationen erkennen, bedenkt man wie der Dotplot aus Tabelle 2.1 mit $ANALYSISSEQUENCEPRIMER$ als Zeilenbeschriftung aussehe.

Das Problem, das sich ergibt, sind nichtrelevante Punkte im Dotplot, welche bei großen Sequenzen den Dotplot in ein einziges Pixelrauschen verwandeln können. Einige solche Punkte waren schon im Beispiel zu erkennen. Um dieses Rauschen heraus zu filtern, gibt es aber Methoden, von denen die nächsten beiden Abschnitte handeln.

2.3 Maizel-Lenk Algorithmus

Dieses Verfahren wurde nach dessen Schöpfern Maizel und Lenk benannt welche es erstmal im Jahre 1981 publizierten [Maizel (1981)]. Es ist ein Verfahren, welches zwei zusätzlicher Variablen bedarf, einmal einer ganzzahligen Fenstergröße und eines ganzzahligen Grenzwertes in Bezug auf diese und die verwendete Substitutionsmatrix. Nach dem Aufbau des Dotplots wird in einer weiteren Phase über jeden Punkt in der Pixelmatrix ein Fenster gelegt mit dem Punkt als Mittelpunkt und ein Pixel wird nur erzeugt, falls die Summe der Werte im Fenster über dem Grenzwert liegt.

Ein wenig formaler hat jeder Punkt ein Fenster, welches eine Teilmenge $F^{i,j}$ der Werte in des Dotplots repräsentiert. Ein Punkt bleibt beim Maizel-Lenk Algorithmus erhalten, falls gilt:

$$\sum_{x \in F^{i,j}} x > \text{Grenzwert} \tag{2.2}$$

Der Algorithmus führt diese Überprüfung dann für alle Punkte des Dotplot durch.

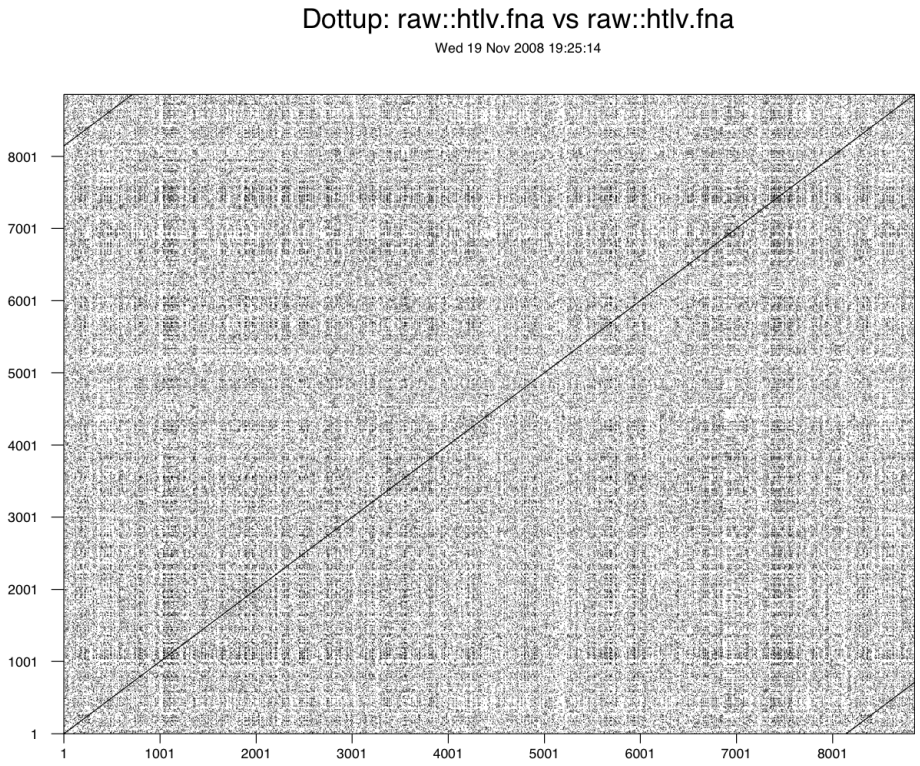


Abbildung 2.1: Dotplot nach Anwendung des Wilbur-Lipman Algorithmus mit Parameter $k = 4$

Das Verfahren entfernt allerdings nicht nur Rauschen aus einem Dotplot, es kann auch dazu führen, das nicht exakte diagonale Übereinstimmungen hervorgehoben werden indem deren Lücken bis zu einer gewissen Größe aufgefüllt werden. Es dauert zwar einige Zeit, selbst heutzutage den Dotplot zu filtern, aber bei passenden Parameterwerten erzeugt der Algorithmus überraschend gute Ergebnisse. Dies ist eines der großen Probleme des Verfahrens, da man zwei Werte durch zeitaufwendiges Experimentieren dahingehend einstellen muss, dass sie gute Ergebnisse liefern.

2.4 Wilbur-Lipman Algorithmus

Wie der Maizel-Lenk Algorithmus wurde das Verfahren nach seinen Schöpfern Wilbur und Lipman benannt und erstmals im Jahre 1983 publiziert [Wilbur (1983)]. Im Vergleich mit dem vorhergehenden Verfahren ist es auf der einen Seite schneller als Maizel-Lenk, aber auf der anderen Seite gehen aus den Dotplots auch mehr Informationen verloren, die im Zweifelsfall relevant sein könnten, abhängig von der Fragestellung. Das Verfahren bedarf allerdings nur eines Parameters anstelle von zweien wie das Verfahren von Maizel und Lenk.

Der Dotplot wird durchlaufen und es verbleiben nur noch Diagonalen in der Pixelmatrix, die eine mindestlänge von k haben, wobei k der besagte Parameter ist. Der Algorithmus entfernt auch waagerechte und senkrechte Linien aus einem Dotplot anders als der Maizel-Lenk Algorithmus, aber man sollte sich bewusst sein, dass Bereiche mit überdurchschnittlich

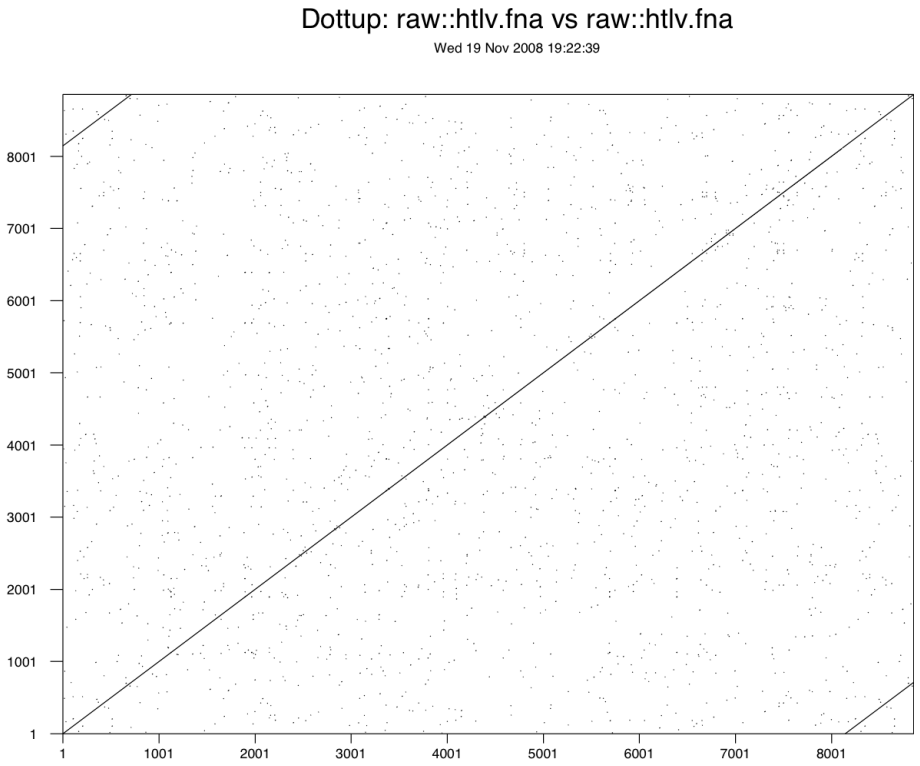


Abbildung 2.2: Dotplot nach Anwendung des Wilbur-Lipman Algorithmus mit Parameter $k = 8$

hohen Teilübereinstimmungen komplett verloren gehen können, falls man z.B. Diagonalen der Länge $k - 1$, wo jeweils zwischen zwei solchen Diagonalen ein fehlender Punkt ist, hat und darauf den Algorithmus anwendet, werden diese Diagonalen kompleet herausgefiltert. Andererseits macht die Tatsache, dass man mit nur einem Parameter arbeitet, die Feineinstellung dessen einfacher. Gut implementiert ist das Verfahren sehr effizient und schnell auch bei großen Sequenzen.

Als Beispiel zum Wilbur-Lipman Algorithmus soll der Vergleich einer Sequenz mit sich selbst dienen, die Sequenz wird mit HTLV bezeichnet, was für *Humanes T-Zell-lymphotropes Virus* steht, und eine ältere Bezeichnung für den HI-Virus ist. Die Beispiele in den Abbildungen 2.1 und 2.2 zeigen die Ergebnisse nachdem der Wilbur-Lipman Algorithmus auf den ursprünglichen Dotplot angewendet wurde.

2.5 Fazit

Es gibt noch 30 Jahre nach der Einführung von Dotplot praktisch relevante Anwendungen für sie. Im Hinblick auf die nachfolgenden Kapitel sei aber nur erwähnt, dass Dotplots eine Hilfestellung dafür sein können wie Bioinformatiker Sequenzen weitergehend vergleichen. Ein Dotplot kann Informationen darüber liefern, ob für eine gegebene Fragestellung ein weiterführender Vergleich mit Hilfe von globalen optimalen Alignments, lokalen optimalen Alignments oder suboptimalen Alignments zu verfolgen ist.

Allerdings muss auch beachtet werden, dass algorithmische wie auch heuristische Ergebnisse keine Informationen über ihre biologische Relevanz enthalten und sie im Regelfall tiefgehender zu überprüfen und zu betrachten sind, bevor aus ihnen Schlussfolgerungen gezogen werden können.

KAPITEL 3

Lokales Sequenz Alignment, beliebige und affine Gap kosten

Ausarbeitung von Olivier Woumpe Dounla, WiSe 2008/09

3.1 Einleitung

Gegenstand dieses Kapitels sind lokal Sequenz Alignment, beliebige und affine Gapkosten. Ein lokales Sequenz Alignment ist ein sehr wichtiges Verfahren in der Bioinformatik, das es ermöglicht die Hervorhebung der Evolution in Proteinfamilie und die Mutationen von Genstrukturen . Die Sequenz-Alignmentsverfahren vergleichen zwei DNA- oder Proteinsequenzen, um ihre Ähnlichkeit zu ermitteln.

Motivation: Viele Gene und Proteine stammen aus einer gemeinsamen Familie mit ähnlicher biochemischen Funktion und/oder gemeinsamer evolutionären Herkunft. Um die Ähnlichkeit von Objekten qualitativ zu erfassen, die Konservierung und die Variabilität zu beobachten, die Verwandtschaft von zwei Sequenzen (Protein oder DNA) zu analysieren, werden solche Sequenz-Alignment-Verfahren benötigt. Da die Biologischen Apparate sehr unterschiedlich sind, können unter gewissen Umständen Insertionen oder Deletionen von Bausteinen sowie Mutationen vorkommen. Die Beobachtung solcher Veränderungen erfolgt zum Beispiel durch Verwendung von in der Bioinformatik(Heuristische und Dynamic programming) vorhandenen algorithmischen Modellen. Heuristische Methode sind zwar sehr schnell, können aber dennoch für manche Sequenzen den *Best-Match* verfehlen.

Problem: "Gesucht wird eine gemeinsame Untereinheit mit der höchsten ähnlichen Region in den vorgegebenen Sequenzen. Gegeben seien z.B. zwei Sequenzen (x und y) und eine Ähnlichkeitsma w. Hier ist eine Teilsequenz $\hat{x}$ von x und $\hat{y}$ von y zu finden, so dass das Alignement im Bezug zu w maximal über alle Teilsequenzspaare ist."

3.2 Dynamisches Programmieren

Unter dynamischer Programmiere versteht man die Lösung eines Problems, indem es in Unterprobleme zerlegt wird und später die besten Ergebnisse verglichen werden. Die in der Bioinformatik meist verwendeten Algorithmen sind Dotplot, Global Alignment und Lokal Alignment. Diese Algorithmen, die seit langem in der Biologie eingeführt und modifiziert wurden, gehören zum Öptimal Alignement“ Algorithmus, da sie die beste Lösung von Optimal Scoring Alignment oder der Menge von Alignment liefern. Man erzeugt mit diesem Verfahren eine Menge von Sequenz Alignment. Als Werkzeug wird eine Matrix verwendet, die sog. Score-Matrix, in der eine Sequenz von Oben nach Unten durchgelaufen wird. Das Ziel hier ist die Berechnung der bestmöglichen Score(Ma für die Ähnlichkeit zweier Symbole) für alle möglichen Positionen in der Matrix, bei Betrachtung aller möglichen Kombinationen von Matches, Mismatches und Gaps an dieser Position. Sehr bekannt sind der *Needleman-Wunsch Algorithmus* und der*Smith- Watermann Algorithmus*, die die Methoden des Sequenzvergleichs erfassen.

- **Scoring-System :** Die Algorithmen allein liefern korrekte Lösungen, aber interpretieren nicht selbst die Symbole, weil unabhängig von der Anwendungsdomäne wird ein Mechanismus benötigt, welches durch die Vorgabe von Scores und die Festlegung der Kosten für die Einführung von Gaps in die Sequenzen, Informationen aus den Anwendungsdomämen übernimmt.

- **Bewertung :** Hier wird zur Bewertung der Ergebnisse eine mathematische Methode(Eng. *log-odds-ratio*) eingesetzt. Wir arbeiten am meisten mit Ähnlichkeitsmatrizen (*Eng. Score Matrices*), die den Logarithmus des Verhältnisses zweier Wahrscheinlichkeiten festgehalten, mit denen Zwei Sequenzen in einem Alignment auftritt. Bekannte Score Matrices sind zum Beispiel die Matrizen aus der PAM-(Position Accepted Mutation) und der BLOSUM-Gruppe(Blocks Substitution Matrix).

3.3 Global Alignment(Needleman- Wunsch Algorithmus)

Beim global Alignment handelt es sich um ein Alignment von zwei oder mehreren Sequenzen, in dem Matches an viele möglichen Stellen auftreten können. Dieses Algorithmus wurde von Needleman und Wunsch geschrieben, und berechnet das optimale Score in die Matrix.

$$F(i,j) = max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$

Hier wollen wir die maximale Anzahl der Matches zwischen zwei Sequenzen und die Lücken (Gaps) am Ende unserer Sequenz finden. Die Erzeugung dieser globalen Sequenzen wird durch eine neue Matrix aufgerufen. Nach der Berechnung des finalen Wertes an der Stelle F(n,m) kann durch Zurückverfolgen(Trace-Back Matrix) der F(i,j)-Einträge, das Alignement abgelesen werden .

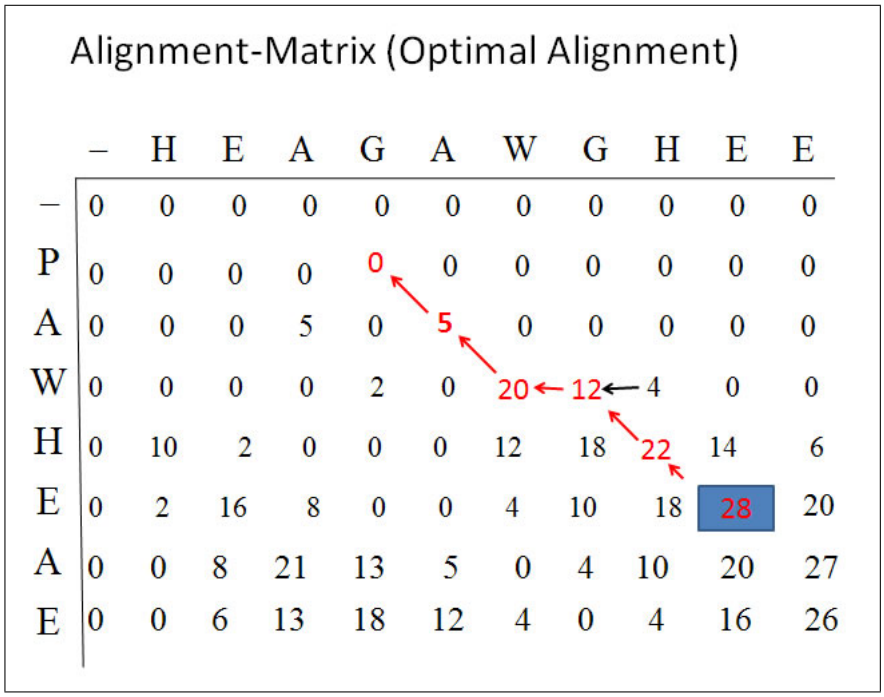


Abbildung 3.1: Alignment-Matrix (Optimal Alignment)

3.4 Lokal Alignment(Smith- Watermann Algorithmus)

Im Vergleich zum Global Alignment wird beim Lokal Alignment nach dem optimalen Teilsequenz Alignement gesucht, deshalb ist es als Verwandt des globalen Alignments dargestellt. Lokal Alignment wird als meist gebrauchte Technik, um Sequenz zu alignieren, da für Sequenzen mit unterschiedlichen Längen, die sich Regionen oder Domänen teilen, Sequenzen mit sehr unterschiedlichen Regionen eignen. Zum Beispiel, die Suche nach einer gemeinsamen Untereinheit vom Proteinsequenzen kann durch lokales Alignment erfolgen, da Proteine oft aus einem Repertoire von ähnlichen Untereinheiten aufgebaut sind, obwohl sie global wenig Ähnlichkeit zeigen. Zum Finden der Teilsequenz wird ein ähnliches Verfahren wie beim globalen Alignment benötigt, das die Berechnung von Scores und die Durchsuchung eines groen Raumes in einer $O(n^2)$ Rechenzeit durchführt. Beim Lokal Alignment Verfahren werden zwei wesentlichen Unterschiede zum Global Alignment betrachtet.

$$F(i,j) = max \begin{cases} 0, \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{cases}$$

1. Alle Werte kleiner als 0 werden durch Nullen ersetzt. Der Grund dafür ist ein neues Alignment zu starten, anstatt das alte auszubauen .
2. Jeder Eintrag der Alingment-Matrix kann als Startpunkt eines Alignments genommen werden. Also anstatt den Wert in der Ecke rechts unten, S(n,m) zu betrachten, wie beim global Alignment erwähnt, suchen wir jetzt nach dem Maximum S(i,j)-Einträge

Abbildung 3.2: Beispiel eines lokalen Alignments Quelle : *Biological sequence analysis*)

in der ganze Matrix, fangen das Trace-Back Prozedure von dort an und fahren fort bis eine Zelle mit dem Wert 0 erreicht wird, was dem Anfang des Alignments entspricht (siehe Abb.)

3.5 Gapkosten

Die Idee hier ist die Einfügung eines Gaps durch ein negatives Score zu bestrafen. Die Bewertung eines Matches oder eines Mismacthes in einer Sequenz wird in das SSoring System"durch positives Score für Matches und negatives Score für Mismatches.

3.5.1 Linear Score:

Die Standard Kosten werden mit einem Gap von Länge g assoziieren und die Kosten für eine Lücke werden proportional zu dessen Länge angenommen. Gap penalties entsprechen ein Wahrscheinlichkeitsmodel von Alignment im Bezug zu die Länge eines Gaps.

$$\gamma(g) = -gd \tag{3.1}$$

3.5.2 Affine Score:

Eine affine Funktion wird hier zur Berechung der Gaps-Penalty benötigt, die Gleichung ist dann :

$$\gamma(g) = -d - (g-1)e \tag{3.2}$$

Wenn die Gap-penalty im Bezug zur Subtitution-Matrix zu teuer sind, wird ein Gap nie im Sequenz Alignment auftauchen.Im Gegenfall werden die Lücken überall im Sequenz Alignment auftauchen.

3.5.3 Alignment mit affinen Gapkosten

Bis jetzt haben wir die Algorithmen betrachtet, wobei die Kosten für ein Gap einfach proportional zu dessen Länge angenommen worden waren. Aber dieses Schema sind keine ideale

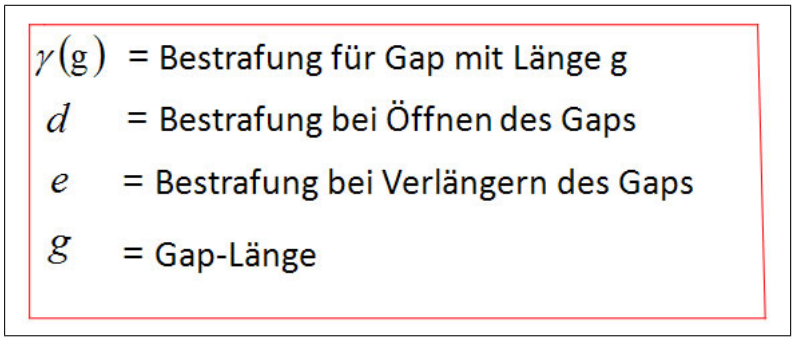


Abbildung 3.3: Gapkosten

Lösung für biologische Sequenzen, da sie bestrafen der Addition von Gap Schritten, Lange Rest von Gap Auftretten und führt zu einer höheren Rechenaufwand. Als verbesserte Methode, verwenden wir hier die affine Gapkosten und jetzt lautet die angepassende Formel:

$$F(i,j) = max \begin{cases} F(i-1,j-1) + s(x_i, y_j), & \\ F(k,j) + \gamma(i-k), & k = 0, ..., i-1 \\ F(i,k) + \gamma(j-k), & k = 0, ..., j-1. \end{cases}$$

Durch diese neue Formel erhöht sich trotz der Verbesserung die Rechenzeit um $O(n^3)$, die jetzt im Hinblick auf die Bestimmung von Alignments optimal ist. Daher werden affinen Gap-Kosten verwendet:
$gamma(g) = -d-(g-1)e$. Daraus ergibt sich nun eine quadratische Rechenzeit $O(n^2)$. Um die neue Rechenzeit zu erreichen, werden drei Möglichkeiten betrachtet: Insertion, Deletion oder Substitution.

$$M(i,j) = max \begin{cases} M(i-1,j-1) + s(x_i, y_j), \\ I_x(i-1,j-1) + s(x_i, y_j), \\ I_y(i-1,j-1) + s(x_i, y_j); \end{cases}$$

$$I_x(i,j) = max \begin{cases} M(i-1,j) - d, \\ I_x(i-1,j) - e; \end{cases}$$

$$I_y(i,j) = max \begin{cases} M(i,j-1) - d, \\ I_y(i-1,j) - e. \end{cases}$$

Hier werden Subtitution und Deletion verwendet:

- $M(i,j)$ für x_i wird mit y_j aligniert,
- $I_x(i,j)$ für x_i mit einem Gap aligniert,
- und I_y für y_j mit einem Gap aligniert.

Ziel dieser Gleichungen ist es, eine Deletion direkt nach einer Insertion zu vermeiden.

KAPITEL 5

Suboptimale Alignments: der Waterman-Eggert Algorithmus

Ausarbeitung von Andrew Quinn

5.1 Einleitung

Nach einer Wiederholung des Smith-Waterman Algorithmus wird der Waterman-Eggert Algorithmus zur Berechnung suboptimaler Alignments betrachtet. Insbesondere werden die nötigen Modifikationen zum Smith-Waterman Algorithmus um solche suboptimale Alignments zu finden diskutiert.

Anwendung In der Untersuchung von Nukleotid-Sequenzen können auch lokale Alignments außer dem optimalen biologisch signifikant sein. Diese sog. suboptimalen Alignments können z. B. weitere Hinweise enthalten über die evolutionäre Übertragung genetischer Information von einem Organismus zu einem anderen. Ferner wird ein sinnvoller Vergleich einer Sequenz mit sich selbst ermöglicht, um evtl. signifikante Muster in ihrer Struktur zu entdecken.

Definitionen Mit einer Sequenz ist in diesem Kapitel eine Folge von Nukleotiden gemeint. Wenn zwei ganze Sequenzen so aneinander angeordnet werden, dass sie nach einer beliebig definierten Scoring-Funktion maximal ähnlich sind, handelt es um ein optimales Alignment. Sollen diese zwei Sequenzen Teilfolgen aus größeren Sequenzen sein, wird dies als lokales Alignment bezeichnet. Alignments, die Scores unter dem in der vorhandenen Daten höchst erreichbaren Wert haben und sich nicht mit dem optimalen Alignment schneiden heißen suboptimale Alignments.

5.2 Der Smith-Waterman Algorithmus

In der Suche nach suboptimalen Alignments ist der Smith-Waterman Algorithmus zur Berechnung optimaler lokaler Alignments ein notwendiger erster Schritt. Erst wenn dieses Alignment gefunden und aus der Datenmenge entfernt worden ist kann nach suboptimalen Alignments gesucht werden.

Um das optimale lokale Alignment zu berechnen wird im Smith-Waterman Algorithmus jedes Nukleotid aus Sequenz a paarweise mit jedem Nukleotid aus Sequenz b verglichen. Die aus der dynamischen Programmierung bekannten Methode der Memosation wird hier angewendet, d.h. diese kleinere Ergebnisse werden abgespeichert um möglicherweise in späteren Schritten neue Berechnungen zu ersparen. Somit wird eine kürzere Rechenzeit gewonnen auf Kosten eines höheren Speicherbedarfs.

Scoring-Funktion Um die Ähnlichkeit zweier Sequenzen $a = a_1a_2...a_n$ und $b = b_1b_2...b_m$ zu bestimmen muss zuerst eine passende Metrik definiert werden. Angenommen $a_i, b_j \in A \cup \{\_\}$, wobei A ein endliches Alphabet und $\_$ eine Lücke, oder Gap, ist, gibt es vier zulässige Situationen, die auftauchen können.

Da hier eine maximale Ähnlichkeit verlangt wird, werden die zulässigen Fälle so bewertet: $s(a_i, b_j)_{a_i=b_j} > 0$, $s(a_i, b_j)_{a_i \neq b_j} < 0$ und $s(a_i, b_j) = -w_k$ bei Lücken, auch Insertions genannt, der Länge k. Solch eine Lücke entsteht wenn a_i oder b_j zum k. aufeinanderfolgenden mal das Gapzeichen ist. Das triviale Alignment $s(\_, \_)$ wird nicht zugelassen, weil es in diesem Kontext ohne Bedeutung ist. Um den Fall $\begin{smallmatrix} a_x \\ - \end{smallmatrix}$ $\begin{smallmatrix} - \\ b_y \end{smallmatrix}$ zu vermeiden, wird w_k so gewählt, dass $2 \cdot |w_1| > |s(a, b)_{a \neq b}$. Damit ist ein Mismatch, d.h. $\begin{smallmatrix} a_x \\ b_y \end{smallmatrix}$ wo $a_x \neq b_y$, bevorzugt über so ein Alignment mit abwechselnden Gaps.

Matrixberechnung Die einzelnen Vergleiche der Nukleotiden werden in einer 2-dimensionalen Matrix abgespeichert. Diese Matrix definieren wir rekursiv nach der folgenden Schema:

$$H_{i,j} = \max\{0; S(a_x a_{x+1}...a_i, b_y b_{y+1}...b_j) : 1 \leq x \leq i, 1 \leq y \leq j\}. \tag{5.1}$$

Die Anfangselemente dieser Scoring-Matrix werden wie folgt definiert:

$$H_{k0} = H_{0l} = 0, 1 \leq i \leq n, 1 \leq j \leq m.$$

Die Gesamtscores der weiteren Elemente werden dann so berechnet:

$$H_{ij} = \max\{0, H_{i-1,j-1} + s(a_i, b_j), E_{ij}, F_{ij}\}, \tag{5.2}$$

wobei für die Matrizen E und F der Gapkosten gilt:

$$E_{ij} = \max_{1 \leq k \leq j}\{H_{i,j-k} - w(k)\} \tag{5.3}$$

$$F_{ij} = \max_{1 \leq k \leq i} \{H_{i-k,j} - w(k)\}. \tag{5.4}$$

Somit wird bei einem Alignment einen Pfad mit aufsteigender Score erzeugt. Wenn die Berechnung der Scoring-Matrix abgeschlossen ist, kann mithilfe des Backtracing Verfahrens das optimale lokale Alignment gefunden werden.

Backtracing Damit der bestmögliche Pfad markiert bleibt, wird der beste Vorgänger in einer Richtungsmatrix gespeichert, wie z.B. beim Dijkstra Algorithmus. Diese Richtungsmatrixelemente $e_{i,j}$ für jedes entsprechende Scoring-Matrix-Elemente $H_{i,j}$ und seinen Vorgänger $H_{u,v}$ sind definiert als

$$e_{i,j} = \begin{cases} -1 & : H_{u,v} = H_{i-1,j} \\ 0 & : H_{u,v} = H_{i-1,j-1} \\ 1 & : H_{u,v} = H_{i,j-1} \end{cases}.$$

Zuletzt sollen Segmente mit Score 0 vom Anfang und Ende des Alignments entfernt werden, damit ein möglichst kompaktes Alignment zurückbleibt. Beim Anfang des Backtraces gilt dann

$$\text{Falls } H_{i,j} = H_{k,l} \text{ und } i + j < k + l, \text{ traceback von } (i, j),$$

und am Ende analog:

$$\text{Falls } H_{p,q} = H_{r,s} \text{ und } r + s < p + q, \text{ traceback von } (p, q).$$

Damit das beste Alignment jetzt aus der Scoring-Matrix ausgelesen werden kann, wird das Element mit der größten Score ausgesucht und dem von der Richtungsmatrix markierten Pfad gefolgt bis die Score des nächsten Elementes gleich Null ist. Bei jedem Knoten des Backtracing-Pfades werden die Werte a_i und b_j ausgegeben. Somit erhält man das optimale lokale Alignment der zwei Sequenzen.

Laufzeitreduktion Gotoh hat beobachtet, dass wenn die Gapkosten als lineare Funktion definiert werden, kann die Laufzeit der Berechnung ein Element der Gapkosten-Matrizen E und F von $O(n)$ auf $O(1)$ reduziert werden kann. Da die Werte $H_{i-1,j-1}$ aus (5.2) schon bekannt sind und somit mit $O(1)$ Zeitaufwand bestimmt werden können, folgt dass die gesamte Laufzeit des Smith-Waterman Algorithmus von der Aufwand der Berechnung von E und F abhängig ist. Weil diese Werte für jedes der maximal n^2 Elemente in H berechnet werden muss, folgt durch diese Reduzierung eine Senkung der Laufzeit der Berechnung der Scoring-Matrix von $O(n^3)$ auf $O(n^2)$.

Satz von Gotoh 1. *Sei* $w(k) = u + v \cdot k$, *so ist* $F_{i,j} = \max\{H_{i-1,j} - w_1, F_{i-1,j} - v\}$.

Beweis.

$$\begin{aligned} F_{ij} &= \max_{1\le k\le i}\{H_{i-k,j} - w_k\} \\ &= \max\{H_{i-1,j} - w_1, \max_{2\le k\le i}(H_{i-k,j} - w_k)\} \\ &= \max\{H_{i-1,j} - w_1, \max_{1\le k\le i-1}(H_{i-1-k,j} - w_{k+1})\} \\ &= \max\{H_{i-1,j} - w_1, \max_{1\le k\le i-1}(H_{i-1-k,j} - w_k) + v\} \\ &= \max\{H_{i-1,j} - w_1, F_{i-1,j} - v\} \end{aligned}$$

□

Da die Werte $H_{i-1,j}$, w_1, $F_{i-1,j}$ und v schon bekannt sind und nicht gesucht werden müssen folgt die Laufzeit von $O(1)$ für Gapkosten-Matrix F. Die Laufzeitreduzierung für E erfolgt analog.

5.3 Der Waterman-Eggert Algorithmus

Durch den Smith-Waterman Algorithmus hat man eine Methode, optimale lokale Alignments zu finden. Der nächste Schritt ist diese Alignment η aus der Scoring-Matrix zu entfernen indem man alle Elemente $H_{i,j} \in \eta$ gleich Null setzt (siehe Tabellen 5.1 und 5.2). Die Alignments die zunächst gefunden werden sind suboptimal. Damit keine verfälschten Werte, die von den entfernten Elementen in der Scoring-Matrix abhängig sind zurückbleiben, muss H erneut berechnet werden. Es wurde allerdings von Waterman und Eggert beobachtet, dass ein gewisser Einflussbereich existiert um das entfernte Alignment herum ab dessen Grenze die Elemente der alten Scoring-Matrix nicht neu berechnet werden müssen.

Um die verfälschten Werte komplett von der Scoring-Matrix zu entfernen, muss man die neuen Elemente H^*_{ij}, E^*_{ij} und F^*_{ij} berechnen bis alle drei der folgenden Bedingungen erfüllt sind:

$$\begin{aligned} \text{I.} \quad & H^*_{ij} = H_{ij}, \\ \text{II.} \quad & E^*_{ij} = E_{ij} \text{ und} \\ \text{III.} \quad & F^*_{ij} = F_{ij} \end{aligned}$$

So können andere Werte in der Matrix nicht mehr von den entfernten Werten beeinflusst werden. Bemerkenswert ist auch, dass nur die Elemente unterhalb und rechts von den entfernten Elementen beeinflusst werden können, und damit auch neu berechnet werden müssen (siehe Tabelle 5.3). Waterman und Eggert haben aus empirischer Erfahrung beobachtet, dass diese Neuberechnung etwa $O(L^2)$ Rechenzeit beträgt, wenn L die Länge des entfernten Alignments ist. Es folgt, dass die gesamt Aufwand um $1 + N$ Alignments zu berechnen liegt bei ca. $O(n^2) + N \cdot O(L^2)$ Rechenzeit.

5.4 Literatur

	G	G	C	T	A	C	T	C	T	A	C	T	G	A	A	C
C	0	0	**10**	0	0	10	0	10	0	0	10	0	0	0	0	10
C	0	0	10	**1**	0	10	1	10	1	0	10	1	0	0	0	10
A	0	0	0	1	**11**	0	1	0	1	11	2	0	0	10	10	0
A	0	0	0	0	11	**2**	0	0	0	11	2	0	0	10	20	1
T	0	0	0	10	0	2	**12**	0	10	0	2	12	0	0	1	11
C	0	0	10	0	1	10	0	**22**	2	1	10	0	3	0	0	11
T	0	0	0	20	0	0	20	2	**32**	12	0	20	0	0	0	0
A	0	0	0	0	30	10	0	11	12	**42**	22	2	11	10	0	0
C	0	0	10	0	10	40	20	0	2	22	**52**	32	12	2	1	20
T	0	0	0	20	0	20	0	0	20	2	32	**62**	42	22	2	0
A	0	0	0	0	30	10	30	41	21	30	12	42	53	52	32	12
C	0	0	10	0	10	40	20	40	32	12	40	22	33	44	43	42
T	0	0	0	20	0	20	50	30	50	30	20	50	30	24	35	34
G	10	10	0	0	11	0	30	41	30	41	21	30	60	40	20	26
C	0	1	20	0	0	21	10	40	32	21	51	31	40	51	31	30
T	0	0	0	30	10	1	31	20	50	30	31	61	41	31	42	22

Tabelle 5.1: vor Entfernung des optimalen Alignments (Smith & Waterman, 1987)

	G	G	C	T	A	C	T	C	T	A	C	T	G	A	A	C
C	0	0	**0***	0	0	10	0	10	0	0	10	0	0	0	0	10
C	0	0	10	**0***	0	10	1	10	1	0	10	1	0	0	0	10
A	0	0	0	1	**0***	0	1	0	1	11	2	0	0	10	10	0
A	0	0	0	0	11	**0***	0	0	0	11	2	0	0	10	20	1
T	0	0	0	10	0	2	**0***	0	10	0	2	12	0	0	1	11
C	0	0	10	0	1	10	0	**0***	2	1	10	0	3	0	0	11
T	0	0	0	20	0	0	20	2	**0***	12	0	20	0	0	0	0
A	0	0	0	0	30	10	0	11	12	**0***	22	2	11	10	0	0
C	0	0	10	0	10	40	20	0	2	22	**0***	32	12	2	1	20
T	0	0	0	20	0	20	0	0	20	2	32	**0***	42	22	2	0
A	0	0	0	0	30	10	30	41	21	30	12	42	53	52	32	12
C	0	0	10	0	10	40	20	40	32	12	40	22	33	44	43	42
T	0	0	0	20	0	20	50	30	50	30	20	50	30	24	35	34
G	10	10	0	0	11	0	30	41	30	41	21	30	60	40	20	26
C	0	1	20	0	0	21	10	40	32	21	51	31	40	51	31	30
T	0	0	0	30	10	1	31	20	50	30	31	61	41	31	42	22

Tabelle 5.2: vor Neuberechnung der Umgebung (Smith & Waterman, 1987)

	G	G	C	T	A	C	T	C	T	A	C	T	G	A	A	C
C	0	0	**0***	**0***	0	10	0	10	0	0	10	0	0	0	0	10
C	0	0	**10***	**0***	**0***	10	1	10	1	0	10	1	0	0	0	10
A	0	0	0	**1***	**0***	**0***	1	0	1	11	2	0	0	10	10	0
A	0	0	0	0	**11***	**0***	**0***	0	0	11	2	0	0	10	20	1
T	0	0	0	10	0	**2***	**0***	**0***	10	0	2	12	0	0	1	11
C	0	0	10	0	1	10	**0***	**0***	**0***	**1***	10	0	3	0	0	11
T	0	0	0	20	0	0	20	**0***	**0***	**0***	**0***	20	0	0	0	0
A	0	0	0	0	30	10	0	**11***	**0***	**0***	**0***	**0***	**11***	10	0	0
C	0	0	10	0	10	40	20	0	**2***	**0***	**0***	**0***	**0***	**2***	1	20
T	0	0	0	20	0	20	0	0	20	**0***	**0***	**0***	**0***	**0***	**0***	**0***
A	0	0	0	0	30	10	30	41	21	**30***	**10***	**0***	**0***	**10***	**10***	**0***
C	0	0	10	0	10	40	20	40	32	12	**40***	**20***	**0***	**0***	**1***	**20***
T	0	0	0	20	0	20	50	30	50	30	20	**50***	**30***	**10***	**0***	**0***
G	10	10	0	0	11	0	30	41	30	41	21	30	60	**40***	**20***	**0***
C	0	1	20	0	0	21	10	40	32	21	51	31	40	51	31	**0***
T	0	0	0	30	10	1	31	20	50	30	31	61	41	31	42	22

Tabelle 5.3: nach Neuberechnung der Umgebung (Smith & Waterman, 1987)

KAPITEL 6

BLAST: Basic Local Alignment Search Tool

Ausarbeitung von Ettiboa Adouakou, WiSe 2008/09

6.1 Einführung

Dieses Kapitel widmet sich BLAST, der weltweit am meisten genutzten Programme zur Analyse biologischer Sequenzdaten. BLAST steht für Basic Local Alignment Search Tool, und wurde entwickelt durch Stephen Altschul, Warren Gish und David Lipman an der *NBCI (National Center for Biotechnology Information)* in der USA.
Der wurde entwickelt damit man die Ähnlichkeit zwischen zwei Sequenzen schnell berechnen kann. Die Aufgabe von BLAST ist dann das Vergleichen von einer experimentell ermittelte DNA- oder Protein-Sequenz mit bereits in einer Datenbank vorhandenen Sequenzen.

Motivation Die Sequenzen Datenbanken wachsen stätig mit meistens DNA dessen funktionen unbekannt sind. Zum Beispiel die GenBank Data enthälte in 2008 etwa 98868465 Sequenzen gegen nur 606 in 1982. Man Braucht daher Computer-Tools um mögliche funktionen diese Sequenzen zu ermitteln.
Die dynamische programierung Methoden erlauben das berechnen von optimal Alignment in linearen Zeit. Verwendet auf einer Sequenzdatenbank würde die berechnungszeit linear zu die groe der Datenbank wachsen. Die Suche wäre erheblich langsam.
BLAST ist eine heuristische Methode. Man selektiert zuerst abschnitt von der Abfragesequenz die einer höhe signifikanz haben können. Danach werden diese mit die Sequenzen in der Datenbank auf Ähnlichkeiten untersucht. Das Vergleich erfolgt auf Basis der Algorithmus von Smith und Watermann.
Durch die Selektion wird die Berechnung von optimal Alignment eingegrenzt und dadurch Zeit gespart. Dies bringt allerdings das Risiko das man nur sub-optimal Alignment erhält.

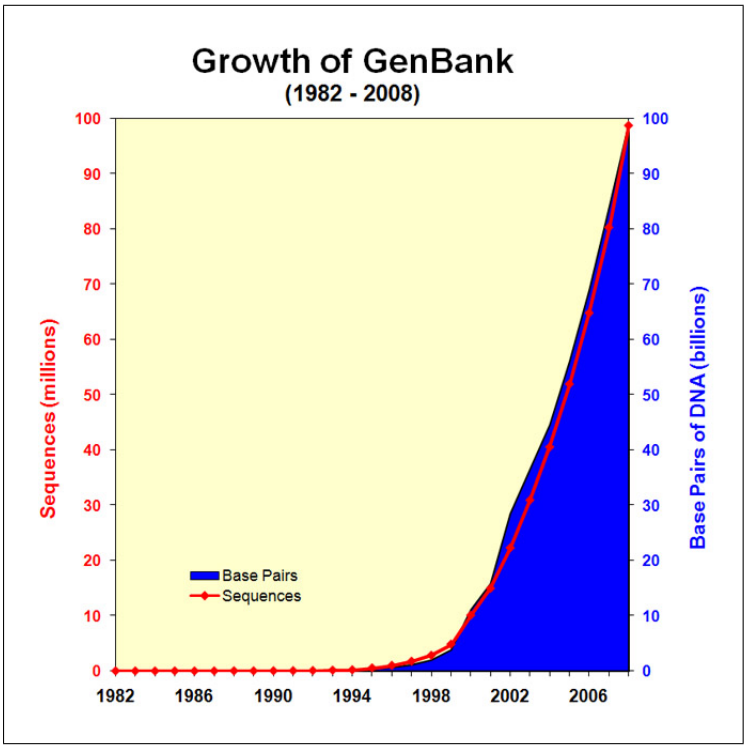


Abbildung 6.1: Quelle: http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html

Die Hauptanwendung von BLAST ist die Suche nach Paralogen und Orthologen Genen und Proteinen innerhalb eines oder mehrerer Organismen. Paralogen sind Verwandte Genen innerhalb eines Genoms, die unterschiedlicher Funktionen haben. Orthologen sind Genen in unterschiedlichen Spezies, die funktional verwandt sind und von einem gemeinsamen Vorläufer abstammen. Im Folgende werden wir das Algorithmus genauer erlautern und seine Variante präsentieren.

6.2 Algorithmus Beschreibung

Aufgabe von BLAST Vergleicht experimentell ermittelte DNA- oder Protein-Sequenzen mit bereits in einer Datenbank vorhandenen Sequenzen.

Eingabe Experimentell ermittelte DNA- oder Protein-Sequenz.

Ergebnis

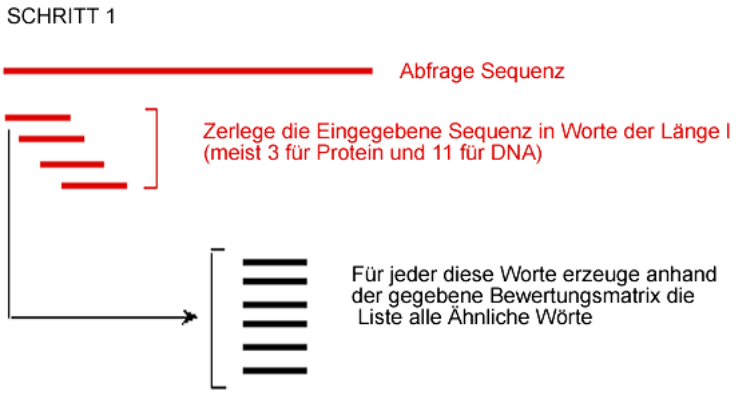
- Ähnlichkeiten zu Sequenzen in der Datenbank (Liste vonLokaler Alignment).
- Signifikanz der Werte.

6.2.1 Vorgehenweise

Die Schritte des Algorithmus sind folgende:

Schritt 1

- Zerlege die Eingegebene Sequenz in Worte der Länge l (meist 3 für Protein und 11 für DNA).
- Für jeder diese Worte erzeuge anhand der gegebene Bewertungsmatrix die Liste alle Ähnliche Worte.
- Worte die eine Bewertung kleiner T erhalten, wobei T ein fixierte Wert ist, werden aus der Liste entfernt.



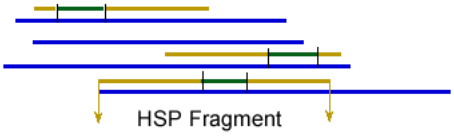
Beispiel Angenommen ist die folgende Abfragesequenz : PQGHLTFYIFQLM, $T = 13$
Man benuzt die BLOSUM62 Matriz. Wir zerlegen dann unsere Abfragesequenz und erhalten folgende Worte: PQG, QGH, GHL, HLT, LTF, ..., QLM. Das erste Wort ist PQG und hat ein Score von 18.
Die Liste der ähnliche Worte ist dann: PEG 15, PKG 14, PRG 14, PQT 13, PSG 13, PTG 13, etc.

Schritt 2

- Suche jedes Wort aus der Liste in Schritt 1 in alle Sequenzen der Datenbank.
- Erzeuge eine Hit-Liste mit die bestimmte position in der Datenbank.

Schritt 3

- Benutze die Sequenzen aus der Hit-Liste als Residuenpaare.
- Versuche jeder Residuenpaar links und rechts zu erweitern.
- Residuenpaar werden hinzugefügt solange der Abstand zwischen die beide Hits kleiner A ist.
- Der langste Fragment wird HSP(High Scoring Pairs) genannt.
- Füge die HSPs die eine Bewertung gröer als S zu einer Liste.



Schritt 4 Berechne die statistische Signifikanz von jeder HSP Berwertung.

Signifikanz von Alignment Nehmen wir an dass wir zwei Sequenzen mit eine sehr gröe Ähnlichkeit haben. Ist diese Ähnlichkeit signifikant oder könnte die zufällig generiert werden?
Wenn der Score ein beobachtete Alignment ist nicht besser als der erwartete score von einer zufällige Permutation der Sequenz ist, dann ist es wahrscheinlich das es sich durch zufall ergegeben hat.

P-Wert Die Wahrscheinlichkeit p dass ein score S zwischen 2 zufällige Sequenzen gröer als ein beobachtete score x (zum Bsp. zwischen ein Abfragesequenz und eine Datenbank) ist defieniert durch:

$$p(S \geq x) = 1 - exp(-e^{-\lambda(x-u)})$$

$u = [log(Km'n')]/\lambda$.
n' ist die länge der Abfragesequenz.
m' ist die länge der ähnliche Sequenz in der Datenbank.

Wenn Der P-Wert sehr klein ist dann ist es unwahrscheinlich das der beobachtete Score sich durch zufall ergeben kann. Daher ist der Alignment von gröer bedeutung. Für Datenbank suche muss noch der E-Wert berechnet werden.

E-Wert Der E-Wert von einem Alignment gibt die erwartete Anzahl der Hits in der Datenbank, deren Score mindestens so gro ist wie der beobachtete.

$$E \approx 1 - e^{-p(S>x)D}$$

Wobei D ist die Anzahl der Sequenzen in der Datenbank.

Die Werte von E sind zwischen 0 und der Anzahl von Sequenzen in der Datenbank. Wie bei der P-Wert, wenn der E-Wert sehr klein ist dann ist es unwahrscheinlich das der beobachtete Score sich durch zufall ergeben kann.

6.2.2 Ausgabe von BLAST

Um BLAST benutzen zu können Muss man ein Standalone Programm installieren oder zu folgender URL: `http://ncbi.org/blast` gehen.

6.3 Variante des Algorithmus

MEGABLAST

- wird empfohlen zur Suche von identischen Sequenzen zu einer eigenen Sequenz.
- wurde speziell erstellt, um besonders lange Sequenzen mit vorhandenen Gegenstücken aus der Datenbank abzugleichen.

BLASTP Vergleicht eine Aminosäuresequenz gegen eine Proteinsequenzdatenbank.

PSI-BLAST Position-Specific Iterative BLAST: Benutzt man, um entfernte Verwandte eines Proteins zu bestimmen. Schritte:

1. erstelle eine Liste aller sehr ähnlichen Proteine.
2. erstelle über diesen Proteinen ein Profil.
3. sende mit diesem Profil erneut eine Suchanfrage an die Proteindatenbank und erhalte eine gröere Gruppe ähnlicher Sequenzen.
4. Mit dieser Gruppe kann zu 2 zurück oder stoppe.

BLASTN Vergleicht eine Nukleotidsequenz gegen eine Nukleotidsequenzdatenbank.

PHI-BLAST Pattern-Hit-Initiated BLAST: funktioniert wie PSI-BLAST. Der Unterschied liegt daran, dass man zuerst nach ein gegebene komplex Muster sucht.

KAPITEL 7

Hidden Markov Modell, Viterbi-, Forward- und Backward-Algorithmus

Ausarbeitung von Manuel Allhoff

7.1 Einleitung

Dieses Kapitel befasst sich mit dem Hidden Markov Modell und dem Viterbi-, Forward- und Backward-Algorithmus.

Dazu wird das Beispiel der CpG-Inseln vorgestellt und in einem Exkurs klar gemacht, wieso die CpG-Inseln wichtig für die Forschung sind. In diesem Zusammenhang wird gleichzeitig die Markov Kette verdeutlicht und mit ihrer Hilfe das HMM erläutert. Mit diesem Wissen werden dann der Viterbi-, der Forward-, und der Backward-Algorithmus erklärt.

7.2 CpG-Inseln

In der Bioinformatik ist es von Interesse, wichtige DNA-Teilstücke, die Promotorbereiche, zu erkennen. Diese Promotorbereiche sind häufig auch sogenannte CpG-Inseln. Sie dienen auf dem DNA-Strang als Markierung, um die Gen-Expression zu ermöglichen.

Bei dem Dinukleotid CG eines DNA Stranges ist das Cytosin häufig methyliert und kann somit relativ häufig in ein T mutieren. Somit tritt ein CG-Paar selten auf und widerspricht der intuitiven Annahme einer Gleichverteilung der Paare. Es gibt allerdings Regionen auf dem DNA-Strang, wo das CG Paar relativ häufig vorkommt. Dies ist wie oben beschrieben z.B. bei Promotorbereichen der Fall. Diese Teilstücke nennt man CpG-Inseln. Man schreibt „CpG“ statt „CG“ um die Bindung von C und G mit einer Wasserstoffbrücke zwischen den Strängen zu unterscheiden. Das Kürzel „CpG“ steht für „Cytosin-phosphatidyl-Guanosin“.

Die CpG-Inseln weisen also eine erhöhte CpG Dichte auf als andere Regionen auf dem DNA-Strang und haben eine Länge von ca. 1-2kb.

Exkurs: Bedeutung der CpG-Inseln Die Bedeutung der CpG-Inseln wird schnell klar, wenn man sie in Verbindung mit Krebserkrankungen bringt. Krebs ist in der Medizin als bösartiger Tumor definiert. Unter einem Tumor versteht man die unkontrollierte Neubildung von Gewebe im Körper. Ein bösärtiger Tumor dringt in das ihn umgebende Gewebe ein und zerstört es. Außerdem kann er sich über das Blut- und Lymphsystem im Körper verteilen und Tochtergeschwülste bilden.

Die unkontrollierte Neubildung von Gewebe ist auf defekte DNA-Teilstücke zurückzuführen, sodass Erbinformationen bei jeder DNA-Kopie verfälscht weitergegeben werden. Jeder Kopiervorgang beginnt bei einem Promotor, der wie oben beschrieben auch häufig eine CpG-Insel darstellt. Somit ist leicht ersichtlich, dass es für die Forschung unerlässlich ist, die CpG-Inseln in einem DNA-Strang zu finden (Lokalisierungsproblem) oder zu entscheiden, ob ein Teilstrang der DNA eine CpG-Insel ist oder nicht (Diskriminationsproblem).

7.3 Markov Kette

Da die CpG-Inseln nur eine größere Wahrscheinlichkeit für das Auftreten eines CG-Dinukleotids im Vergleich zum restlichen DNA-Strang beschreiben, kann man bei einem gegebenen DNA-Teilstück auch nicht sagen, ob es sich entweder um eine CpG-Insel handelt oder nicht. Stattdessen muss man eine Wahrscheinlichkeit festlegen, die beschreibt, wie groß die Chance ist, dass dieses Teilstück eine CpG-Insel ist oder nicht. Erst dann kann man eine Entscheidung treffen.

Somit kann man das Diskriminations-Problem wie folgt definieren:

Problem 1: Diskriminations-Problem

Gegeben:
Eine Zeichenkette x mit der Länge L über dem Alphabet $Z = \{A, C, G, T\}$

Gesucht:
Entscheidung, ob x eine CpG-Insel ist oder nicht.

Um dieses Problem zu lösen, muss man ein Modell finden, dass DNA-Sequenzen generieren kann, die entweder CpG-Inseln sind oder nicht. Die passenden Wahrscheinlichkeiten, z.B., dass in einer CpG-Insel ein G auf ein C folgt, sind empirisch aus DNA-Sequenzen gewonnen. Dazu hat man über eine lange Sequenz die Nukleotide gezählt und anschließend verschiedene Teilstücke miteinander verglichen. Erst mit diesem Verfahren hat man entdeckt, dass es so etwas wie CpG-Inseln überhaupt gibt.

Ein passendes Modell, dass diese Anforderungen erfüllt, ist die Markov Kette, die formal wie folgt definiert ist:

Definition 1. *Eine Markovsche Kette erster Ordnung (kurz: Markovsche Kette) ist ein stochastischer Prozess $X_1, X_2 \ldots$ mit Werten in einer höchstens abzählbaren Zustandsmenge Z, der die Markovsche Eigenschaft besitzt:*

Für jeden Zeitpunkt $t > 0$ und jeder Zustandsfolge $z_1, z_2, \ldots, z_t$ mit

$P(X_1 = z_1, X_2 = z_2, \ldots, X_{t-1} = z_{t-1}) > 0$ *ist*

$P(X_t = z_t | X_1 = z_1, X_2 = z_2, \ldots, X_{t-1} = z_{t-1}) = P(X_t = z_t | X_{t-1} = z_{t-1})$.

Die Wahrscheinlichkeiten $P(X_1 = s) = a_{Bs}$ heißen Startwahrscheinlichkeiten, die Wahrscheinlichkeiten $a_{st} = P(X_i = t | X_{i-1} = s)$ Übergangswahrscheinlichkeiten.

Eine Markov Kette kann man als Graph darstellen, wie man in Abbildung 7.1 sehen kann.

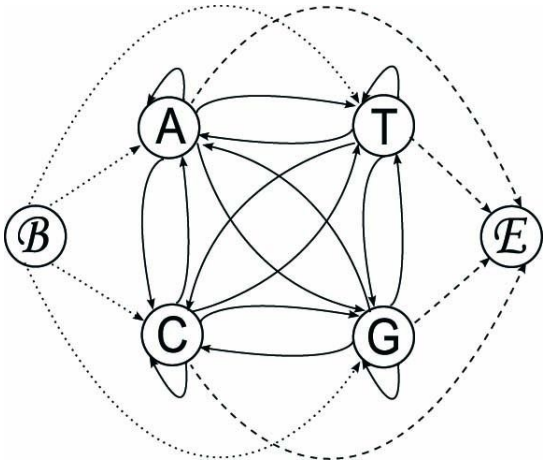


Abbildung 7.1: DNA-Markov Kette als Graph, angelehnt an: (Durbin et al., 1998a, S. 50)

Grundlagen Mit dieser Markov Kette kann man eine DNA-Sequenz generieren, indem man sich einen Pfad von B nach E aussucht. Für die Zustandsmenge gilt $Z = \{A, C, G, T\} \cup \{B, E\}$, wenn die Markov-Kette beim Erreichen von E endet. Wenn die Markov-Kette nach einer festen Anzahl an Schritten endet, gilt: $Z = \{A, C, G, T\} \cup \{B\}$. Die Kanten entsprechen den Übergangswahrscheinlichkeiten, die durch

$$a_{st} = P(x_i = t | x_{i-1} = s)$$

bestimmt sind. Dabei entsprechen s und t Zuständen aus Z und x_i bzw. x_{i-1} einem Symbol aus der DNA-Sequenz. Die Übergangswahrscheinlichkeit a_{st} gibt also die Wahrscheinlichkeit an, dass dem Zustand s der Zustand t folgt.

Als Spezialfall ist hier zum einen die *Startwahrscheinlichkeit* a_{Bs} zu nennen. Sie gibt an bei welchem Symbol die Sequenz beginnt und es gilt:

$$a_{Bs} = P(x_1 = s).$$

Zum anderen die *Endwahrscheinlichkeit*, für die gilt: $a_{tE} = P(x_{L+1} = E | x_L = t)$, $\forall\ L \geq 0$. Sie modelliert das Ende der Sequenz. Beide werden oft aus Gründen der Übersichtlichkeit einfach weggelassen.

Für jede Sequenz $x = (x_L, x_{L-1}, \ldots, x_1)$ fester Länge L kann man ihre Wahrscheinlichkeit wie folgt angeben:

$$P(x) = P(x_L, x_{L-1}, \ldots, x_1).$$

Diese Formel kann man durch mehrmalige Anwendung der Regel $P(X, Y) = P(X|Y)P(Y)$ umformen zu

$$P(x) = P(x_L|x_{L-1}, \ldots, x_1)P(x_{L-1}|x_{L-2}, \ldots, x_1) \ldots P(x_1).$$

Laut Definition der Markov Kette hängt der aktuelle Zustand nur von seinem Vorgängerzustand ab, und somit gilt:

$$P(x) = P(x_L|x_{L-1})P(x_{L-1}|x_{L-2}) \ldots P(x_2|x_1)P(x_1).$$

Wenn man dabei den Start- und Endzustand außer Acht lässt, was oft passiert, ist die Formel äquivalent zu:

$$P(x) = P(x_1) \prod_{i=2}^{L} a_{x_{i-1}x_i}.$$

Lösung des Diskriminations-Problems Mit diesem Wissen kann man zwei Markov Ketten erstellen, die entweder eine CpG-Insel (im folgenden „+ -Modell“ genannt), oder keine CpG-Insel („- -Modell“) generieren. Die wie oben beschrieben empirisch gewonnenen Übergangswahrscheinlichkeiten sind durch die Formel

$$a_{st}^{+} = \frac{c_{st}^{+}}{\sum_{t'} c_{st'}^{+}}$$

bestimmt. Beim „- -Modell“ erfolgt die Berechnung analog. Die Variable c_{st}^{+} gibt an, wie oft von dem Zustand s in den Zustand t gewechselt wird. Dieser Wert wird anschließend durch die Summe aller Übergänge dividiert. So erhält man folgende Tabelle:

+	A	C	G	T	-	A	C	G	T
A	0,180	0,274	0,426	0,120	A	0,300	0,205	0,285	0,210
C	0,171	0,368	0,274	0,188	C	0,322	0,298	0,078	0,302
G	0,161	0,339	0,375	0,125	G	0,248	0,246	0,298	0,208
T	0,079	0,355	0,384	0,182	T	0,177	0,239	0,292	0,292

Tabelle 7.1: Übergangswahrscheinlichkeiten innerhalb und außerhalb von CpG-Inseln, Quelle: (Durbin et al., 1998a, S. 51)

Man definiere nun eine Score-Funktion S:

$$S(x) = \log \frac{P(x|+)}{P(x|-)}$$

Sie berechnet den Quotienten von den Wahrscheinlichkeiten, dass die Zeichenkette x eine CpG-Insel ist bzw. nicht ist.

Es gilt:

$$S(x) = \sum_{i=1}^{L} \log \frac{a_{x_{i-1}x_i}^{+}}{a_{x_{i-1}x_i}^{-}},$$

da man für jeden String $x = (x_1, \ldots, x_L)$ mit fester Länge L den Quotienten abschnittsweise berechnet und die Werte addiert.

Man rechnet mit der log-Funktion, da man durch mehrmaliges Multiplizieren von Wahrscheinlichkeiten sehr schnell sehr kleine Zahlen als Ergebnis erhält, die bei der digitalen Berechnung zu einer Underflow-Exception führen. Die Funktion S ist intuitiv vernünftig gewählt, da eine CpG-Insel im „+ -Modell“ eine größere Wahrscheinlichkeit hat als im „- -Modell“. Der Score wird also positiv. Der umgekehrte Fall gilt analog. Somit gilt:

$$S(x) = \begin{cases} > 0, \text{ wenn } x \text{ CpG-Insel,} \\ < 0, \text{ wenn } x \text{ keine CpG-Insel,} \\ = 0, \text{ keine Aussage.} \end{cases}$$

7.4 Hidden Markov Modell

Definition 2. *Ein Hidden Markov Modell ist ein Quintupel $\lambda = (S, A, \pi_{Start}, B, V)$ mit:*

- *endlicher Menge S von Zuständen $\{s_1, \ldots, s_n\}$*
- *Matrix von Übergangswahrscheinlichkeiten $A = \Big(a_{ij} | a_{ij} = P(s_t = j | s_{t-1} = i)\Big)$*
- *Vektor π_{Start} von Zustandsstartwahrscheinlichkeiten*
- *Ausgabealphabet V*
- *Emissionswahrscheinlichkeiten $e_k(b) = P(x_i = b | \pi_i = k)$ in Form einer Matrix E mit $k \in S$ und $b \in V$*

Grundlagen In der bis jetzt beschriebenen Thematik kann man ein HMM als eine erweiterte Markov Kette betrachten. Neben dem Zufallsexperiment der einzelnen Zustandsübergänge wird in jedem Zustand ein weiteres Experiment durchgeführt: Die Auswahl des Emissionssymbols b, das durch die Matrix der Emissionswahrscheinlichkeiten E gegeben ist. Bei einem HMM wird also das Symbol b von dem Zustand k entkoppelt und es gilt:

$$e_k(b) = P(x_i = b | \pi_i = k)$$

Dabei beschreibt π die Sequenz der Zustände. Der Übergang von einem Zustand zum anderen ist durch folgende Formel bestimmt:

$$a_{kl} = P(\pi_i = l | \pi_{i-1} = k)$$

Die gemeinsame Wahrscheinlichkeit einer Symbolsequenz x und einer Zustandsfolge π mit fester Länge L lautet

$$P(x, \pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}.$$

Zuerst wird also der Startzustand durch $a_{0\pi_1}$ bestimmt. Dann wird für jedes einzele Symbol das Produkt aus der Emissionswahrscheinlichkeit und der Übergangswahrscheinlichkeit bestimmt.

Hier wird der Unterschied zu einer Markov Kette deutlich, der auch das Wort „hidden“ in HMM rechtfertigt. Das einzelne Symbol b ist nicht eindeutig durch einen Zustand bestimmt, da es in mehreren Zuständen möglich ist b zu generien. Der Pfad π ist also nicht eindeutig.

Beispiel Um die Definition zu verdeutlichen, wird diese anhand des Beispiels des „unehrlichen Casinos“ erklärt.

Gegeben sei ein Casino, das zeitweise einen gezinkten Würfel benutzt, sodass die Gleichverteilung der Zahlen 1 bis 6 nicht mehr gegeben ist. Bei dem gezinkten Würfel fällt die 6 mit einer Wahrscheinlichkeit von 1/2, alle anderen Zahlen mit einer Wahrscheinlichkeit von 1/10. Der Croupier wechselt nach jedem Wurf mit einer gewissen Wahrscheinlichkeit den Würfel oder fährt mit dem aktuellen fort. Diese Situation kann man als HMM modellieren, das sich wie die Markov Kette als Graph darstellen lässt (vgl. Abbildung 7.2).

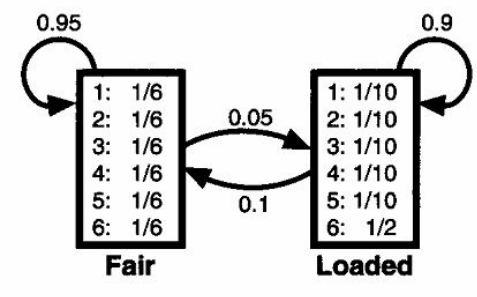


Abbildung 7.2: HMM als Graph: Das unehrliche Casino, Quelle: (Durbin et al., 1998a, S. 55)

Hierbei gilt:

- $S = \{1, 2\}$ mit 1=Fair und 2=Loaded
- $V = \{1, 2, 3, 4, 5, 6\}$
- $\pi_{Start} = (\frac{1}{2}, \frac{1}{2})$
- $A = \begin{pmatrix} 0.95 & 0.05 \\ 0.1 & 0.9 \end{pmatrix}$
- $E = \begin{pmatrix} 1/6 & 1/10 \\ 1/6 & 1/10 \\ 1/6 & 1/10 \\ 1/6 & 1/10 \\ 1/6 & 1/10 \\ 1/6 & 1/2 \end{pmatrix}$

Lokalisierungsproblem HMMs haben gegenüber den Markov Ketten den Vorteil, dass sie DNA-Sequenzen generieren, in denen CpG-Inseln auftreten können, aber nicht müssen. Das passende HMM ist in Abbildung 7.3 dargestellt. Hierbei kommen noch die Übergangswahrscheinlichkeiten innerhalb der „+“ - und „-“ -Modelle hinzu, wie es bei Abbildung 7.1 der Fall ist.

Das HMM wird sich aufgrund der Übergangswahrscheinlichkeiten wahrscheinlich öfter im „- -Modell“ befinden. Das ist intuitiv klar, da die CpG-Inseln in einer DNA-Sequenz auch selten vorkommen. Es stellt sich die Frage, wo diese Inseln in der gegebenen DNA-Sequenz sind. Dies wird als Lokalisierungs-Problem bezeichnet.

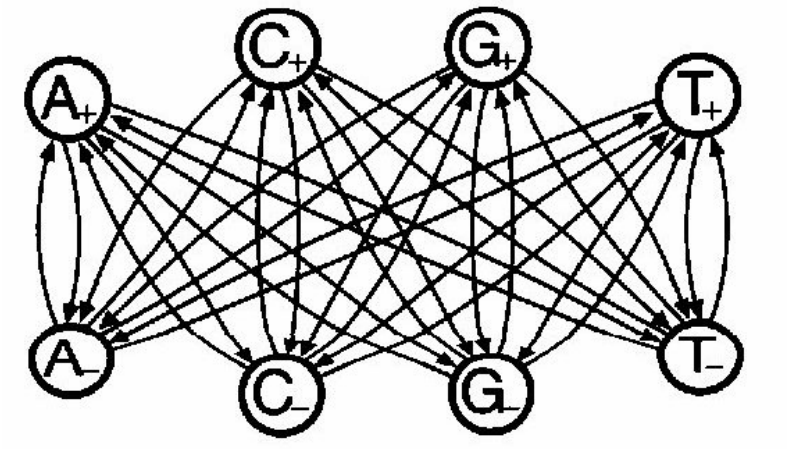


Abbildung 7.3: DNA HMM, Quelle: (Durbin et al., 1998a, S. 53)

Problem 2: Lokalisierungs-Problem

Gegeben:
Eine Zeichenkette x mit der Länge L über dem Alphabet $Z = \{A, C, G, T\}$

Gesucht:
Position der CpG-Inseln

Anmerkung zum Lokalisierungs-Problem: Die Lösung, die vorgestellt wird, erklärt nicht, wie genau die Position angegeben wird. Dies hängt von der konkreten Implementierung ab, auf die hier nicht eingegangen wird.

Eine Lösung des Problems ist, es auf das Diskriminations-Problem zu reduzieren. Man legt ein Fenster der Größe l Nukleotide fest und betrachtet die DNA-Sequenz als Eingabe-Stream. Dieser Stream wird nun Nukleotid für Nukleotid in das Fenster eingegeben und bei jedem Schritt als neues Diskriminations-Problem betrachtet. Bei dieser Lösung ist aber nicht klar, wie l gewählt werden muss, um sinnvolle Ergebnisse zu erhalten.

Stattdessen kann man bei einer gegebenen Zeichenkette x den wahrscheinlichsten Pfad durch das HMM in Abbildung 7.3 suchen. Dieser kann durch das „+ -Modell“ führen, und somit hat man die CpG-Insel(n) lokalisiert. Dieses Problem löst der Viterbi-Algorithmus.

7.5 Viterbi-Algorithmus

Um den Viterbi-Algorithmus zu erklären, bietet sich eine andere Darstellungsart von HMMs an: Die „Irrfahrt“. Mit einer „Irrfahrt“ kann man eine zeitliche Abfolge der Zustandsübergänge simulieren. Jede Spalte entspricht dabei einem Zeitpunkt. Die Kanten geben weiterhin die Übergangswahrscheinlichkeiten an. Abbildung 7.4 zeigt die „Irrfahrt“ für das zeitweise „unehrliche Casino“.

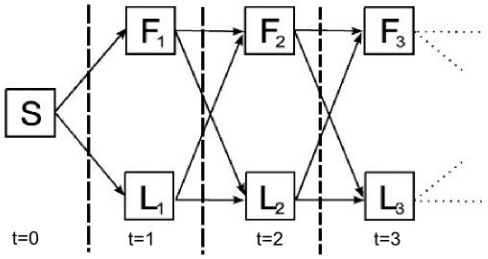


Abbildung 7.4: unehrliches Casino, Darstellung als „Irrfahrt“

Der Viterbi-Algorithmus arbeitet in drei Phasen, der Initialisierungs-, der Pfadberechnungs-, und der Backtrackingsphase. Dabei werden 2 Datenstrukturen verwendet. Für jeden Zustand in der „Irrfahrt“ wird ein Wert abgespeichert, der angibt, wie wahrscheinlich es ist, zum Zeitpunkt t in diesem Zustand i zu sein. Diese lokale Pfadwahrscheinlichkeit ist durch

$$\delta_t(i) = \max_{\pi} P(x_1, \ldots, x_t, \pi_1, \ldots, \pi_t) \text{ mit } \pi_t = i$$

gegeben. Weiter wird für jeden Zustand i ein Zeiger abgespeichert, der auf den Vorgängerzustand $i-1$ zeigt: $\phi_k(i)$.

Initialisierung Bei der Initialisierung wird die lokale Pfadwahrscheinlichkeit für den Zeitpunkt $t = 1$ berechnet. Der Zustand bei $t = 0$ wird immer mit einer Wahrscheinlichkeit von 1 eingenommen. Die Pfadwahrscheinlichkeit ergibt sich bei einer Eingabesequenz $x = (x_1, \ldots, x_L)$ aus dem Produkt der Emissionswahrscheinlichkeit $e_t(x_i)$ und der Zustandsstartwahrscheinlichkeit π_i:

$$\delta_1(i) = e_i(x_1)\pi_i$$

Pfadberechnung Während der Pfadberechnungsphase wird für jeden Zeitpunkt und Zustand eine lokale Pfadwahrscheinlichkeit berechnet. Es gilt:

$$\delta_t(i) = \max_j \left(\delta_{t-1}(j) a_{ji} e_i(x_t)\right)$$

Die neu zu berechnende Pfadwahrscheinlichkeit $\delta_t(i)$ ergibt sich aus dem Produkt der Pfadwahrscheinlichkeit des vorangegangenen Zustandes, die Übergangswahrscheinlichkeit von diesem zu dem aktuellen Zustand und der Emissionswahrscheinlichkeit des passenden Symbols des aktuellen Zustandes. Da es mehrere Pfade zu dem aktuellen Zustand geben kann, wird das Maximum als neuer Wert gewählt. Der Zustand, mit dessen Hilfe man diesen maximalen Wert berechnet hat, wird als Vorgängerzustand in $\phi_t(i)$ eingetragen.

Backtrackingsphase Wenn für jeden Zeitpunkt und Zuständ die Pfadwahrscheinlichkeit berechnet wurde, wird für den letzten Zeitpunk $t = L$ der Zustand i_L mit der maximalen Wahrscheinlichkeit ermittelt. Mit Hilfe von $\phi_L(i_L)$ kann der wahrscheinlichste Pfad von diesem Zustand bestimmt werden. Die umgekehrte Ausgabe dieser Vorgängerzustände entspricht dem gesuchten wahrscheinlichsten Pfad.

Laufzeit Zur Laufzeitbestimmung sind nur die Schlüsselvergleiche bei der Pfadberechnung zu beachten, da die anderen Berechnungen des Algorithmus in konstanter Zeit geschehen können. Man kann sich intuitiv an der Abbildung 7.5 die Laufzeit des Algorithmus verdeutlichen. Gegeben sei das Alphabet $Z = \{A, B, C\}$ und eine Sequenz der Länge L. Zum Zeitpunkt $k = 1$ werden $|Z|$ Schlüsselvergleiche, und zum Zeitpunkt $k = 2, k = 3 \ldots |Z^2|$ benötigt. Insgesamt werden also $(L-1)|Z|^2 + |Z|$ Schlüsselvergleiche vorgenommen. Und diese liegen in $O(L|Z|^2)$.

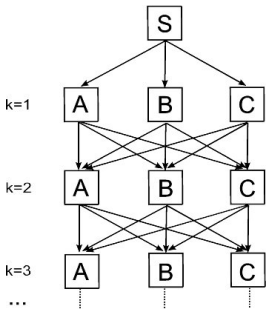


Abbildung 7.5: Laufzeit des Viterbi-Algorithmus

7.6 Forward-Algorithmus

Der Forward-Algorithmus berechnet die Wahrscheinlichkeit, dass ein HMM zum Zeitpunkt t in einem Zustand k ist und die Sequenz $x = (x_1, x_2, \ldots, x_t)$ erzeugt hat (Evaluierungsproblem I).

Problem 3: Evaluierungs-Problem I

Gegeben:
Eine Beobachtung $x = (x_1, x_2, \ldots, x_t, \ldots, x_L)$ über einem Alphabet und ein HMM δ

Gesucht:
Wahrscheinlichkeit, dass das HMM die Sequenz $(x_1, \ldots, x_t)$ erzeugt und im Zustand k ist.

Der Algorithmus hat eine Initialisierungs-, Rekursions-, und Terminationsphase. Zudem wird die Vorwärtsvariable

$$f_k(i) = P(x_1, \ldots, x_i, \pi_i = k)$$

berechnet. Sie gibt die Wahrscheinlichkeit an, dass die Sequenz $(x_1, \ldots, x_i)$ generiert wurde und sich das HMM im Zustand $\pi_i = k$ befindet. Sie ist rekursiv durch

$$f_l(i+1) = e_l(x_{i+1}) \sum_k f_k(i) a_{kl}$$

definiert. Dazu wird in der Initialisierungsphase $f_S(i)$ festgelegt.

Dabei ist e_l die Wahrscheinlichkeit, das passende Symbol auszugeben. Der Term $f_k(i)a_{kl}$ beschreibt die Wahrscheinlichkeit, sich im Zustand k, der zeitlich vor l liegt, zu befinden, und die passende Ausgabe bis zu diesem Punkt generiert zu haben. Dies wird mit der Übergangswahrscheinlichkeit zu dem aktuellen Zustand l multipliziert, um die Gesamtwahrscheinlichkeit zu erhalten. Da mehrere Kanten in einer „Irrfahrt“ nach l führen können, werden die Wahrscheinlichkeiten addiert. Wenn alle $f_l(i)$ festgelegt sind, wird die Gesamtwahrscheinlichkeit für die Sequenz x durch

$$P(x) = \sum_k f_k(L)a_{k0}$$

berechnet, die für den Forward-Backward-Algorithmus wichtig ist.

7.7 Backward-Algorithmus

Im Gegensatz zum Forward-Algorithmus berechnet der Backward-Algorithmus nicht die Wahrscheinlichkeit in einen Zustand k zu gelangen. Er geht davon aus in dem Zustand k zu sein und berechnet die Wahrscheinlichkeit eine (Teil-)Sequenz zu erzeugen (Evaluierungsproblem II).

Problem 4: Evaluierungs-Problem II

Gegeben:
Eine Beobachtung $x = (x_1, x_2, \ldots, x_t, \ldots, x_L)$ über einem Alphabet und ein HMM δ

Gesucht:
Wahrscheinlichkeit, von einem Zustand k aus $(x_{t+1}, \ldots, x_L)$ zu erzeugen.

Dazu wird die Rückwärtsvariable $b_k(x_i)$ definiert:

$$b_k(i) = P(x_{i+1}, \ldots, x_L | \pi_i = k)$$

Sie gibt genau die gesuchte Wahrscheinlichkeit an: Die Sequenz $(x_{i+1}, \ldots, x_L)$ soll erzeugt werden unter der Bedingung, dass sich das HMM zu Anfang im Zustand k befand.

Wie die Vorwärtsvariable, so ist auch die Rückwärtsvariable rekursiv definiert:

$$b_k(i) = \sum_l a_{kl} e_l(x_{i+1}) b_l(i+1)$$

Die Initialisierung erfolgt durch $b_k(L) = a_{k0}$. Somit kann man $b_k(i)$ für jeden Zustand und Zeitpunkt berechnen. Dazu ermittelt man das Produkt aus der schon bekannten Rückwärtsvariable, der Emissionswahrscheinlichkeit und der Übergangswahrscheinlichkeit zu dem aktuellen Zustand. Da sich in einer „Irrfahrt“ so ebenfalls mehrere Pfade ergeben können, werden die einzelnen Wahrscheinlichkeiten addiert.

7.8 Forward-Backward-Algorithmus

Der Forward-Backward-Algorithmus kombiniert -wie der Name schon sagt- die letzten beiden vorgestellten Algorithmen. Mit seiner Hilfe kann man die Wahrscheinlichkeit berechnen, mit der sich das HMM bei einer gegebenen Sequenz x im Zustand k befindet:

Problem 5: Evaluierungs-Problem III

Gegeben:
Eine Beobachtung $x = (x_1, x_2, \ldots, x_t, \ldots, x_L)$ über einem Alphabet und ein HMM δ

Gesucht:
$P(\pi_i = k|x)$, d.h. die Wahrscheinlichkeit, dass sich das HMM bei einer gegebenen Beobachtung x im Zustand k befindet.

Die Wahrscheinlichkeit, dass x generiert wird und sich das HMM im Zustand k befindet, kann man somit wie folgt in einem Ausdruck zusammenfassen:

$$P(x, \pi_i = k)$$

Man kann die Sequenz x der Länge L auch als $(x_1, \ldots, x_i, x_{i+1}, \ldots, x_L)$ schreiben und somit ergibt sich der äquivalente Ausdruck:

$$P(x_1, \ldots, x_i, x_{i+1}, \ldots, x_L, \pi_i = k)$$

Mit Hilfe der Regel $P(A, B) = P(A|B)P(B)$ formt man mit $A = x_{i+1}, \ldots, x_L$ und $B = x_1, \ldots, x_i, \pi_i = k$ den Ausdruck um:

$$P(x_{i+1}, \ldots, x_L|x_1, \ldots, x_i, \pi_i = k)P(x_1, \ldots, x_i, \pi_i = k)$$

Da nach der Definition der Markov Kette der aktuelle Zustand nur von seinem Vorgängerzustand abhängt, erhält man:

$$P(x_{i+1}, \ldots, x_L|\pi_i = k)P(x_1, \ldots, x_i, \pi_i = k)$$

Dies ist nach Definition äquivalent zu

$$b_k(i)f_k(i)$$

Die im Evaluierungsproblem III gesuchte Wahrscheinlichkeit $P(\pi_i = k|x)$, ergibt sich aus dem Quotienten dieser Formel und der Wahrscheinlichkeit von x, die man mit dem Forward-Algorithmus berechnen kann. Es gilt:

$$P(\pi_i = k|x) = \frac{P(x, \pi_i = k)}{P(x)} = \frac{f_k(x_i)b_k(x_i)}{P(x)}$$

Somit löst der Forward-Backward-Algorithmus das Evaluierungsproblem III.

KAPITEL 8

HMM Training

Ausarbeitung von Domagoj Jakulj, WiSe 2009

8.1 Einleitung

Während sich das voriges Kapitel den Anwendungen und der Auswertung bereits vorhandener *Hidden Markov Modelle* widmet, konzentrieren wir uns in diesem Kapitel auf das Finden von sinnvollen Parametern für *HMMs* mit Hilfe von Trainingsdaten.

Anwendungen Die Hauptaufgabe von *HMMs* ist die Modellierung von Systemen, deren zugrundeliegende Bedeutungen und Zustände nicht direkt beobachtbar sind. Sie werden sowohl im Feld der Spracherkennung, als auch in der Bioinformatik ausgiebig verwendet, um von beobachtbaren Daten (Audio-Signale, DNA-Sequenzen) auf Interpretationen dieser Daten (Phoneme/Wörter, Promoter-Region) zu schliessen. Nun stellt sich sehr schnell die Frage, wie man ohne Wissen über die genaue Beschaffenheit der zugrundeliegenden Zustände Modelle erstellen kann, die diese repräsentieren, oder zumindest in guter Näherung wiedergeben.

Dass *HMMs* als Heuristiken dienen, um Aussagen zu treffen, ob eine isolierte und beobachtete DNA-Sequenz Teil einer CpG-Insel und daher wahrscheinlich auch Teil einer Promoter-Region ist, haben wir im vorherigen Kapitel gesehen. Die Güte dieser Heuristiken ist offensichtlich stark davon abhängig, wie die Zustände und Parameter dieser *HMMs* gewählt wurden.

Eine kleine Auswahl an Methoden, um zumindest die Parameter solcher *HMMs* abzuschätzen und zu *trainieren*, wird dieses Kapitel behandeln und den Baum-Welch-Algorithmus als eine dieser Methoden im Detail vorstellen.

Definitionen *Hidden Markov Modelle* definieren wir im folgenden Kapitel als ein Quadtupel aus Zustandmenge Z, einem Ausgabealphabet Σ, der Zustandsübergangsfunktion a_{kl}, sowie der Emissionsfunktion $e_k(b)$, mit $k, l \in Z$ und $b \in \Sigma$.

$$\Theta = \{Z, \Sigma, a_{kl}, e_k(b)\}$$

Hierbei liefert die Zustandsübergangsfunktion die jeweilige Wahrscheinlichkeit des *HMMs* von Zustand k in Zustand l zu wechseln. Ähnlich liefert die Emissionsfunktion $e_k(b)$ die Wahrscheinlichkeit, Zeichen b im Zustand k zu emittieren.

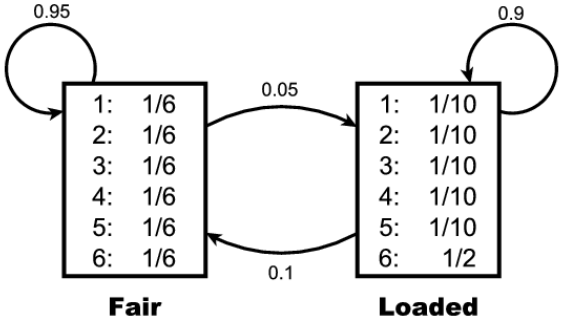


Abbildung 8.1: Ein Unfaires Casino als *HMM*. vgl.:(Durbin et al., 1998b, Seite 55)

8.2 Vorabbetrachtung

Eines der Hauptmerkmale, welche *HMMs* von herkömmlichen *Markov-Ketten* unterscheiden, ist genau der Umstand, das die Zustandsfolge, die zu einer beobachteten Zeichensequenz gehört, unbekannt ist. Ist der Pfad durch ein *HMM* hingegen bekannt, ist es ein Leichtes, die Parameter für dieses *HMM* so zu wählen, das die Wahrscheinlichkeit für diese Trainingsfolge maximiert wird.

Maximum Likelihood Estimators Für die Wahrscheinlichkeiten der Zustandswechsel von Zustand k nach Zustand l ist dies offensichtlich die Anzahl der beobachteten Zustandswechsel von k nach l, geteilt durch die Anzahl aller Zustandswechsel von k. (Wobei wir den Verbleib im Zustand k als Zustandswechsel von k nach k definieren.)

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \tag{8.1}$$

Analog berechnet man die Wahrscheinlichkeiten der emittierten Zeichen mit:

$$e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')} \tag{8.2}$$

Die beobachtbare Anzahl der Ereignisse, dass in Zustand k Zeichen b emittiert wurde, geteilt durch die Anzahl der Ereignisse, dass Zustand k ein beliebiges Zeichen $b' \in \Sigma$ emittiert.

Die Verwendung dieser beiden *Maximum Likelihood Estimators* (MLE) zur Parameterschätzung für *Markov-Modelle* liefert nur dann gute Näherungen, wenn die Trainingssequenzen eine ausreichende Länge aufweisen. Die Wichtigkeit dessen wird uns sofort anhand des Unfairen Casino-Beispiels klar: Betrachten wir eine Trainingssequenz, in der nur Fünfen und Sechsen auftauchen, und wenden wir nun die beiden *MLEs* auf diese Sequenz an, so bekommen wir für die emittierte Anzahl der Zahlen Eins bis Vier, in beliebigen Zuständen, Werte gleich Null. Dies wiederum führt zu Wahrscheinlichkeiten gleich Null. Es sollte offensichtlich sein das ein Modell, welches weder mit einem fairen noch mit einem unfairen Würfel Zahlen von Eins bis Vier generiert, nicht unseren Erwartungen entspricht.

Analog verhält es sich, wenn wir eine Trainingsfolge wählen, die nur faire Würfelwurfe beinhaltet. In diesem Fall werden sogar sowohl Zähler, als auch Nenner der Formel 8.1 Null, und daher undefiniert.

Pseudocounts Um solche Probleme zu vermeiden, oder um *a priori* Wissen in das Modell und die Wahrscheinlichkeiten mit einfließen zu lassen, bedient man sich sogenannter *Pseudocounts*. Diese werden den tatsächlich beobachteten Zahlen hinzuaddiert. So kann man in unserem Beispiel allen Zahlen zumindest ein Vorkommen garantieren, oder mit entsprechend großen *Pseudocounts* für die Zahlen des fairen Würfels eine Wahrscheinlichkeitsverteilung nahe der Geichverteilung erzwingen.

8.3 Parameterschätzung ohne Pfadkenntnisse

In aller Regel liegen uns aber keine Informationen darüber vor, welche Zeichen von welchem Zustand erzeugt wurden, d. h. keine Informationen über den Pfad durch das *HMM*, der unsere Trainingssequenz repräsentiert. Infolgedessen bleibt uns nichts anderes übrig, als alle möglichen Pfade durch das *HMM* zu betrachten und abzuschätzen, mit welcher Häufigkeit wir Zustandswechsel und die Emission unserer Zeichen in diesen Zuständen erwarten.

log-likelihood Um nun ein *HMM* zu trainieren, müssen wir die Wahrscheinlichkeit, diese Trainingssequenzen mit diesem *HMM* zu generieren, maximieren. Dies bedeutet: Für eine Menge von Trainingssequenzen $\{x^1, x^1, \ldots, x^n\}$, von denen wir annehmen, sie seien unabhängig, ist die Wahrscheinlichkeit gleich dem Produkt der Wahrscheinlichkeiten jeder einzelnen Sequenz.

$$P(x^1, x^1, \ldots, x^n|\Theta) = \prod_{j=1}^{n} P(x^j|\Theta) \tag{8.3}$$

Wahrscheinlichkeiten liegen alle im Intervall 0 – 1, daher werden die Werte für die Multiplikation mehrerer Sequenzen sehr schnell sehr klein. Es bietet sich daher an, den Logarithmus der Wahrscheinlichkeiten zu betrachten. Damit wird dieses Produkt zur Summe der logarithmischen Wahrscheinlichkeiten (*log-likelihood*).

$$\log P(x^1, x^1, \ldots, x^n|\Theta) = \sum_{j=1}^{n} \log P(x^j|\Theta) \tag{8.4}$$

Gesucht wird also ein *HMM*,bei welchem

$$\Theta' = \text{argmax}_\Theta \sum_{j=1}^{n} \log P(x^j|\Theta) \tag{8.5}$$

gilt.

Die Lösung dieser Gleichung ist allerdings ein NP-schweres Problem[1] und wir benötigen daher eine Heuristik, um stattdessen eine Näherung zu berechnen.

Baum-Welch Algorithmus Der *Baum-Welch Algorithmus* stellt – neben dem Viterbi-Training – eine dieser Heuristiken da. Dabei bedient sich der Algorithmus sowohl des *Forward-Algorithmus*, als auch des *Backward-Algorithmus*. Die Wahrscheinlichkeit, dass in einer Trainingssequenz x and Position i von Zustand k nach l gewechselt wird (d. h. : a_{kl} benutzt wird) ist

$$P(\pi_i = k, \pi_{i+1} = l|x, \Theta) = \frac{f_k(i) \cdot a_{kl} \cdot e_l(x_{x+1}) \cdot b_l(i+1)}{P(x)}. \tag{8.6}$$

Von dieser Wahrscheinlichkeit lässt die zu erwartende Anzahl der Verwendungen von a_{kl} herleiten, in dem man über alle Positionen und alle Trainingssequenzen summiert.

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) \cdot a_{kl} \cdot e_l(x_{i+1}^j) \cdot b_l^j(i+1) \tag{8.7}$$

Hierbei ist $f_k^j(i)$ gleich der *Vorwärts-Variable* des Forward-Algorithmus (Algorithmus 1) aus dem vorhergehendem Kapitel. Analog ist $b_l^j(i)$ die *Rückwärts-Variable* des Backward-Algorithmus (Algorithmus 2).

Informell ausgedrückt, summiert man gewichtet über alle möglichen Pfade und errechnet damit die erwarteten Anzahlen der Übergänge und Zeichen-Emissionen für jede Trainingsfolge. Im Detail berechnet die *Vorwärts-Variable* die Wahrscheinlichkeit, sich in Zustand k zum Zeitpunk i zu befinden, unter der Voraussetzung das man bereits $i-1$ Zeichen der Trainingssequenz generiert hat. Dieses multipliziert man mit der Wahrscheinlichkeit, von diesem Zustand k in einen Zustand l zu wechseln und mit der Wahrscheinlichkeit das nächste Zeichen der Trainingsfolge in diesem Zustand l auch zu emittieren. Als letztes wird mit Hilfe der *Rückwärts-Variablen* nun die Wahrscheinlichkeit berechnet, auch den verbleibenden Rest der Trainingsfolge, startend in Zustand l, zu generieren. Abschließend wird die Summe über alle Positionen i gewichtet durch die Gesamt-Wahrscheinlichkeit der Trainingsfolge.

Ähnlich gestaltet sich die Berechnung der erwarteten Anzahlen der emittierten Zeichen:

$$E_k(b) = \sum_j \frac{1}{P(x^j)} \sum_{\{i|x_i^j=b\}} j_k^j(i) \cdot b_k^j(i). \tag{8.8}$$

Hierbei betrachtet die innere Summe nur die Positionen i der Trainingsfolge, in denen das Zeichen b auch ausgegeben wird.

Mit diesen Werten können wir nun neue Parameter für das *HMM* berechnen. Es lässt sich zeigen, das die *log-likelihood* nach jeder Iteration zumindest nicht abnimmt, und in sich der

[1] (Merkl and Waack, 2003b, S. 287)

Tat einem lokalem Maximum nähert. Somit ist es notwendig, den *Baum-Welch Algorithmus* mehrfach mit jeweils neuen, zufälligen Anfangsparametern zu starten. Erst wenn die *log-likelihood* sich auch zwischen diesen separaten Durchläufen nicht signifikant ändert, kann man mit ausreichender Sicherheit davon ausgehen, das man eine Näherung errechnet hat, die nahe beim globalen Maximum liegt. Eine kompaktere Darstellung des Algorithmus (angelehnt an (Durbin et al., 1998b, S. 65) ist im Anschluss als Algorithmus 3 auf Seite 42 zu finden.

Viterbi-Training Eine der Alternativen zum *Baum-Welch Algorithmus* ist das Viterbi-Training, das sich dem Viterbi-Algorithmus aus dem vorherigen Kapitel bedient. Kurz skizziert, berechnet dieses Verfahren nicht sämtliche möglichen Pfade und Wahrscheinlichkeiten, sonder wählt für jede Trainingssequenz jeweils den wahrscheinlichsten Pfad aus, und verwendet diesen zu Schätzung der Parameter des *HMMs*. Daher maximiert das Viterbi-Training auch nicht die *log-likelihood* des *HMMs*, sondern streng genommen nur die Wahrscheinlichkeiten auf den Pfaden, die ohnehin als am wahrscheinlichsten erkannt wurden. Alternative Pfade mit geringerer Wahrscheinlichkeit für jede Trainingssequenz werden nicht weiter betrachtet. Im Allgemeinen Fall zeigt der *Baum-Welch Algorithmus* bessere Ergebnisse als das Viterbi-Training (Durbin et al., 1998b, S. 66).

Initialisierung ($i = 0$): $f_0(0) = 1, f_k(0) = 0$ für $k > 0$

Rekursion ($i = 1 \dots L$): $f_l(i) = e_l(x_i) \sum_k f_k(i-1) \cdot a_{kl}$

Terminierung: $P(x) = \sum_k f_k(L) \cdot a_{k0}$

Algorithmus 1: Forward-Algorithmus

Initialisierung ($i = L$): $b_k(L) = a_{k0}$ für alle k

Rekursion ($i = L-1, \dots, 1$): $b_k(i) = \sum_l a_{kl} \cdot e_l(x_{i+1}) \cdot b_l(i+1)$

Terminierung: $P(x) = \sum_l a_{0l} \cdot e_l(x_1) \cdot b_l(1)$

Algorithmus 2: Backward-Algorithmus

Eingabe Ein *HMM* $\Theta = \{Z, \Sigma.a_{kl}, e_k(b)\}$ mit festgelegter Struktur/Zuständen, *Pseudocounts* A_{kl} und $E_k(b)$. Zudem Trainingsequenzen $X^j = \{x_1^j, x_2^j, \ldots, x_i^j\}$.

Algorithmus Berechne:

1. Initialisierung
   a) Weise willkürlich/zufällig gewählte Parameter den Funktionen a und e zu.
   b) Setze die Abbruchkonstante ϵ auf einen (kleinen) positiven Wert.
2. Iteration
   a) Für jede Trainingsequenz $j = 1, \ldots, n$:
      i. Berechne $f_k(i)$ für Sequenz j mit Hilfe des *Forward-Algorithmus* (Algorithmus 1, S. 41).
      ii. Berechne $b_k(i)$ für Sequenz j mit Hilfe des *Backward-Algorithmus* (Algorithmus 2, S. 41).
      iii. Addiere die gewonnen Werte zu A (8.7) und E (8.8) hinzu.
   b) Berechnung der neuen Modell Parameter mit Hilfe von (8.1) und (8.2)
   c) Berechnung der neuen *log-likelihood*.
3. Terminierung
   a) Speichere alten *log-likelihood* Wert.
   b) Berechnen neuen *log-likelihood* Wert (8.4).
   c) Subtrahiere den alten *log-likelihood* Wert vom neuen. Breche die Berechnung ab, falls das Ergebniss kleiner oder gleich ϵ ist.

Ausgabe Ein *HMM* $\Theta' = \{Z, \Sigma.a'_{kl}, e'_k(b)\}$ mit Parametern für die Zustandsübergangs- und Emissions-Funktion, die eine Verbesserung der *log likelihoods* bezüglich der Trainingssequenzen X^j bedingen.

Algorithmus 3: Baum-Welch Algorithmus

KAPITEL 10

Perfekte Phylogenien

Ausarbeitung von Kada Benadjemia

10.1 Einfuehrung und Motivation

Taxonomie in der Biologie. Die Taxonomie beschäftigt sich mit der Einteilung von Lebewesen in Zusammengehörigkeitsgruppen. Die Gruppen werden als Taxa bezeichnet, ein einzelnes Lebewesen ist ein Taxon. Es existieren verschiedene Ansätze Taxa zu klassifizieren. Die klassische Taxonomie orientiert sich maßgeblich an essentiellen Merkmalen und gilt als unwissenschaftlich. Eine klassische Taxaeinteilung wäre z.B. Säugetiere, Lurche und Vögel usw. Die Abbildung 10.1 links zeigt eine klassische Taxonomie, wie wir sie vielleicht aus der Schule kennen.

Ein modernerer Ansatz ist die kladistische Taxonomien, die als wissenschaftlich angesehen wird. Die kladistische Taxonomie bedient sich der Phylogenese und wird als gewurzelter Baum dargestellt (Kladiogramm, Abbildung 10.1 rechts). Die Phylogenese bezeichnet die stammesgeschichtliche Entwicklung, die auf genetische Merkmale beruht. Eine kladistische Taxonomie stellt also letztendlich die stammesgeschichtliche Entwicklung (phylogenetische Verwandtschaft) der Taxa dar. Die Elternknoten können als Benennung der Zusammengehörigkeitsgruppen dienen.

10.2 Grundbegriffe und Problematik

Phylogenien Eine Phylogenie ist solch ein gewurzelter Baum mit der stammesgeschichtlichen Entwicklung einer Taxagruppe, wobei die Blätter jeweils ein Taxon darstellen und Elternknoten unbeschriftet bleiben. Abbildung 10.2 zeigt einige Taxa und eine dazu passende Phylogenie. Verzweigungen repräsentieren die Entstehung eines Merkmals. Die Kanten bis hin zu einem Blatt(Taxon) weisen alle Merkmale auf, die ein Taxon besitzt. Die Elternknoten

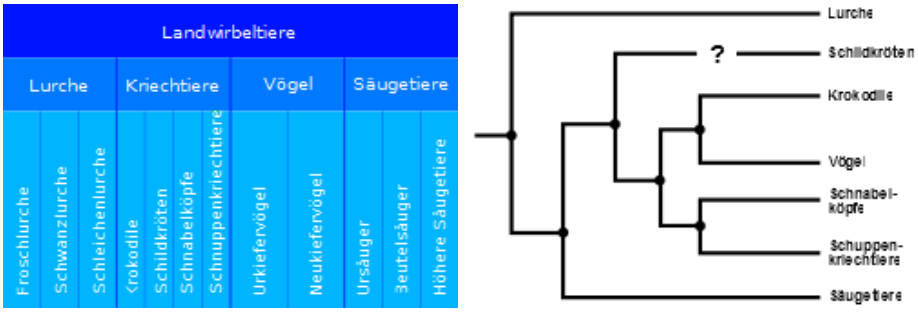


Abbildung 10.1: Links - Einteilung nach klassischer Taxonomie. Rechts - Einteilung nach kladistischer Taxonomie. (Abbildungen aus Wikipedia (b))

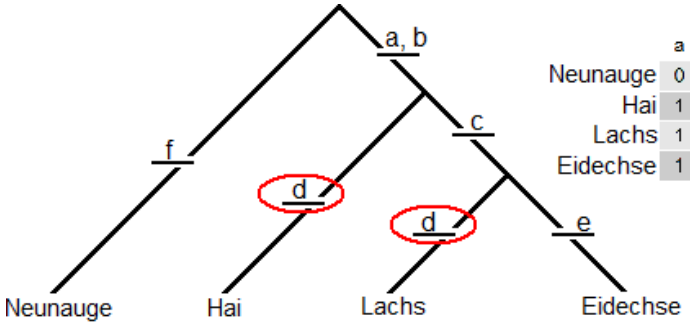
können hier als hypothetische Vorfahren gesehen werden. Als Merkmal können genetische oder anatomische Merkmal herangezogen werden.

Bei der Erstellung einer Phylogie müssen einige Dinge beachtet werden, damit ein Stammbaum zustande kommt der auch realistisch ist.

Maximale Parsimonie Zu beachten ist, dass Erbgutmutationen die Leben zulassen extrem unwahrscheinlich sind. Abgesehen davon sorgen zahlreiche biologische Mechanismen dafür, dass so viele Mutationen wie möglich behoben werden. Die Mutation ist in der modernen Evolutionslehre dennoch das einzige Ereignis das Vielfalt ermöglicht. Demnach muss eine wissenschaftlich akzeptierbare Phylogenie so mutationnssparsam wie möglich sein. Wir wollen also einen Stammbaum erstellen, der so wenig Merkmalsentstehung wie möglich zulässt. Von Nöten ist also eine Phylogenie mit **maximaler Parsimonie**(Sparsamkeit).

Homoplasie Ein weiteres Problem ist die Homoplasie. Homoplasie beschreibt die Entstehung eines bestimmten Merkmals bei mehreren verschiedenen Taxa mit unterschiedlicher stammesgeschichtlichen Entwicklungen. Ein bekanntes Beispiel dafür sind die Flossen beim Wal und beim Hai. Beide Tiere stammen dennoch von völlig verschiedenen Vorfahren ab. Diesen Effekt gibt es auch auf genetischer Ebene. Wenn man echte Lebewesen betrachtet sind Homoplasien in den Daten daher nicht vermeidbar. Homoplasie ist ein schlechter Hinweis für evolutionäre Verwandtschaft, weil Merkmalsgleichheit nicht unbedingt auch Verwandtschaft bedeutet. Wir beschränken uns daher auf auf homoplasiefreie Phylogenien.

Das Perfekte-Phylogenien-Problem Um das Perfekte-Phylogenien-Problem nun näher zu spezifizieren:
Gegeben ist: Eine Menge von Taxa und eine Menge von Merkmalen. Merkmale haben einen Zustand aus dem Merkmalsalphabet
Gesucht ist: Eine homoplasiefreie Phylogenie mit maximaler Parsimonie.



	a	b	c	d	e	f
Neunauge	0	0	0	0	0	1
Hai	1	1	0	1	0	0
Lachs	1	1	1	1	0	0
Eidechse	1	1	1	0	1	0

Abbildung 10.2: Eine Phylogenie für eine Menge von Taxa und Merkmalen. Die dargestellten Merkmale sind: (a) Flossenpaar, (b) Gebiss, (c) large dermal bones, (d) Rückenflosse, (e) Lunge, (f) rasping tungue. Merkmal (d) weist Homoplasie auf. Basierend auf dem Beispiel von Fernandez-Baca.

10.3 Darstellungen und Datenstrukturen

Um das perfekte Phylogenienproblem algorithmisch lösen zu können werden hier folgende Darstellungen und Datenstrukturen besprochen:

Matrizen Die Menge der Merkmale kann als Matrix darsgestellt werden. In einer Matrix m[i,j] ist Zeile i das Taxon i und Spalte j das Merkmal j. Dabei können die Einträge m[i,j] binäre oder n-näre Merkmalszustände sein.
Ein Beispiel für einen Zustand von m[i,j] wäre, dass Taxon i das Merkmal j besitzt. Für diesen Fall wäre die Merkmalszustandsmenge oder das Merkmalsalphabet binär. Die Tabelle in Abbildung 10.2 wäre ein Beispiel für solch eine Matrix.

Gewurzelte Bäume Das Naheliegenste ist es wohl Phylogenien direkt in die Datenstruktur der gewurzelten Bäume zu überführen.
Zwei geläufige Darstellungen sind die mit elternbeschrifteten Knoten und ohne elternbeschrifteten Knoten. In der Darstellung mit elternbeschrifteten Knoten besitzen die Elterknotenkinder alle Merkmale, die ein Elternknoten besitzt. In der Darstellung ohne elternbeschrifteten Knoten besitzen nur Blätter Merkmale.
In beiden Fällen stellen Blätter Taxa dar, Elternknoten sind hypothetische Vorfahren und Kanten bedeuten eine Merkmalsentstehung (Abbildung 10.2).

Graphen Bäume sind nur spezielle Graphen. Der Unterschied zwischen der Darstellung als Baum und der als Graph ist, dass hier keine Wurzel existiert.
Bei Graphen gibt es ebenfall eine die Darstellung mit Elternknotenbeschriftetung und ohne. Hier gilt wieder, dass Blätter Taxa, Elternknoten hypothetische Vorfahren und Kanten Merkmalsentstehungen darstellen.
Ein solcher Graphen zeigt sich in Abbildung 10.3.
Jeder Graph mit und ohne Elternknotenbeschriftung kann zu einem gewurzelten Baum überführt werden, eine beliebige Kante zur Wurzel ernennt. Die Wurzel zeigt in dem Fall auf

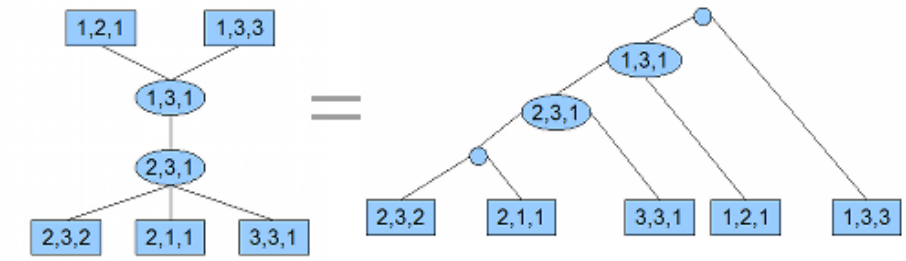


Abbildung 10.3: Phylogeniedarstellung als Graph

die beiden inzidenten Knoten der gewählen Kante (Abbildung 10.3).
Der Vorteil gegenüber kladistischen/gewurzelten Bäumen ist, dass wir unser Phylogenienproblem als graphentheoretisches Problem formulieren und bereits gefundene Lösungen wiederverwenden können.

10.4 Steinerbaeume in Phylogenien

10.4.1 Formulierung als graphentheoretisches Problem

Wir betrachten elternbeschrifteten Graphen.
Die Blätter des Graphen stellen Taxa dar, die Elternknoten hypothetische Vorfahren.
Die Kanten sind gewichtet und stellen nach wie vor Merkmalswechsel dar.
Das Gewicht einer Kante ist die Anzahl an Merkmalen, in denen sich zwei verbundene Knoten unterscheiden; anders ausgedrückt, die Hammingdistanz.

Wir sind immer noch an minimalen Merkmalswechsel interessiert und brauchen dazu die leichteste Verbindung zwischen den Taxa, evtl über hypothetischen Vorfahren.
Genau hier begegnen wir wieder dem Begriff der *maximalen Parsimonie*.
Das *maximale Parsimonienproblem* für Graphen ist besitzt innerhalb des *Steinerbaummproblems* einen Lösungsansatz.

10.4.2 Steinerbaeume

Ein Steinerbaum beschreibt den kleinsten Teilgraphen innerhalb eines umgebenden Graphen, welcher eine Menge an gegebener Terminalknoten miteinander verbindet. Ein Beispiel für einen Steinerbaum lässt sich Abbildung 10.4 betrachten.

Definition Sei G = (V, E) ein zusammenhängender ungerichteter gewichteter Graph ohne Mehrfachkanten, wobei V die Knotenmenge und E die Kantenmenge sei. Des Weiteren sei Terminalmenge T $\subseteq$ V. Das Kantengewicht sei c: E $\rightarrow$ **R**.

Gesucht ist ein minimaler Spannbaum $T_G*(T)$ = (V', E') mit T $\subseteq$ V' $\subseteq$ V, E' $\subseteq$ E.

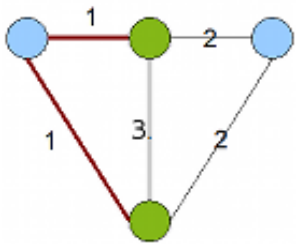


Abbildung 10.4: Steinerbaum. Grüne Knoten sind Terminale, blaue Nichtterminale. Die roten Kanten gehören zum Steinerbaum.

10.4.3 Ueberfuehrung auf das perfekte Phylogenienproblem

Soweit haben wir den Begriff des Steinerbaumproblems geklärt, jedoch ist der Zusammenhang mit Phylogenien noch nicht ganz klar. In einem Baum sind unsere Taxa Blätter, also deklarieren wir die *Taxa* als *Terminale*. Die *Nichtterminale* sollen hier als *hypothetischen Vorfahren*, also als Elternknoten dienen. Um das Steinerbaumproblem auf das perfekte Phylogenienproblem anwenden zu können, gilt es zu klären, woher wir die *Nichtterminale* nehmen.

Hier gibt es verschiedene Ansätze.

- Wir bilden alle Merkmalskombinationsmöglichkeiten aus dem Merkmalsalphabet.
  Für z Merkmalszustände und m Merkmale bilden wir also z^mKnoten.
  Da alle Kombinationen gebildet werden sind unsere Taxa ebenfalls in dieser Knotenmenge enthalten. Verbunden werden aber *nur* Knoten, die sich um genau ein Merkmal unterscheiden. Wir brauchen die Taxa nur noch als Terminale zu deklarieren. Somit bildet der Pfad von einem Terminal zum Anderen (evtl über Nichtterminale) die Hammingdistanz zwischen zwei Taxa.
  Vorteil dieser Variante ist ihre Einfachheit, ihr Nachteil aber ist die gigantische Knotenanzahl. Für binäre Zustände, wenige Merkmale und sehr vielen Taxa entstehen bei dieser Variante jedoch akzeptabel mehr Knoten als bei der Folgenden.
- Wir bilden gezielt für all unsere Taxa Nachbarknoten, die sich um genau ein Merkmal unterscheiden. Somit gibt es bei m Merkmalen mit z Merkmalszuständen für jedes Taxon genau $(z-1)m$ Nachbarn. Also bei t Taxa maximal $(z-1)mt$ Knoten im Graphen. Es werden alle Knoten miteinander verbunden und das Gewicht einer Kante ist wieder die Hammingdistanz.
  Vorteil dieser Variante ist die wesentlich geringere Knotenanzahl im Graphen für n-näre Zustandsalphabeta und viele Merkmale.

Beide Varianten sind äquivalent, bezogen auf das Gewicht für den Pfad zwischen zwei Taxa. Man kann gut nachvollziehen, dass man in der zweiten Variante, *jeden* Pfad in kleinere Teilpfade mit Kantengewicht = 1 unterteilen kann. Es kommen Knoten hinzu die zwischen den Teilpfaden liegen und sich um genau ein Merkmal von ihren Nachbarn unterscheiden. Damit sind getreu der ersten Variante ausschließlich Knoten miteinander verbunden, die eine Hammingdistanz gleich eins besitzen.

Da die Hammingdistanz unsere Merkmalswechsel darstellen und Steinerbäume ein minimale Spannbäume sind, haben wir durch die Berechnung eines Steinerbaums die gewünschte

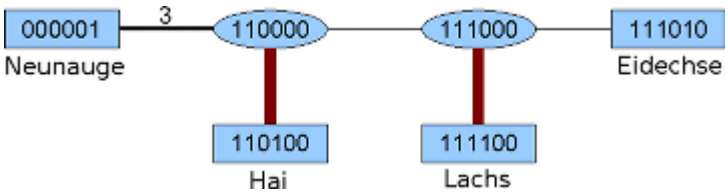


Abbildung 10.5: Graphendarstellung der Phylogenie in Abbildung 10.2. Rotmarkierte Kante stellt die Homologie dar. Das vierte Mermal(d) wechselt einmal zu oft und somit in einen redundanten Zustand.

maximale Parsimonie erreicht.

Nun habe wir zwar eine Phylogenie mit *maximaler Parsimonie*, jedoch ist sie nicht zwangsweise *homoplasiefrei* und damit nicht zwangsweise eine perfekte Phylogenie.
Um sicher zu sein, dass wir eine perfekte Phylogenie besitzen brauchen wir ein Kriterium das eine Homoplasie ausschließt.

10.4.4 Ausschluss von Homoplasie

Um Homomplasie erkennen zu können benötigen wir vorher noch den Begriff der Länge einer Phylogenie:

Definition *Die Länge einer elternbeschrifteten Phylogenie ist die Summe all ihrer Kantengewichte.*

Satz 1. *Eine Taxamenge besitzt genau dann ein perfekte Phylogenie, wenn die minimale Länge ihrer elternbeschrifteten Phylogenie gleich* $\sum_{m \in M} (z_m - 1)$ *ist.*

Mit $z_m - 1$ ist die Anzahl der Zustände gemeint, in die das Merkmal m wechseln kann.
Um etwas Licht ins formale Dunkel zu bringen, hier noch mal eine andere Formulierung:

Formulierung 1. *Die Anzahl der Kanten bei denen ein bestimmtes Merkmal seinen Zustand wechselt, darf nicht großer sein als die Moglichkeiten wiederholungslos zwischen Merkmalszuständen zu wechseln.*

Mit wiederholungslos ist gemeint, dass ein Merkmal niemals zurück in einen bereits eingenommenen Zustand wechseln darf. Die Phylogenielänge wird genau dann größer als die oben genannte Summe, wenn ein Merkmal zu oft gewechselt hat. Nämlich für z Zustände mindestens z mal.
Als Beispiel kann man binäre Zustände heranziehen. Man kann einen binären Zustand genau ein mal wechseln, beim zweiten mal ist man wieder in einen bereits eingenommenen Zustand gewechselt. Man kann sich für binäre Zustände leicht klarmachen, warum solch ein Wechsel Homoplasie bedeutet:
Entsteht ein Merkmal in einer Verzweigung von einem bestimmten Taxon aus, so hat nur eines der Nachfahren dieses Merkmal und das andere nicht. Besitzt der Nachfahre ohne das Merkmal selbst einen Nachfahre bei dem das Merkmal entsteht, so haben beide Taxa mit

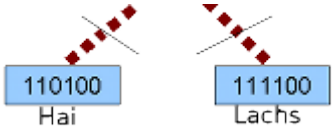


Abbildung 10.6: Teilbäume für den Graphen aus Abbildung 10.5 für Merkmal d = 1

dem Merkmal nicht den gemeinsamen Vorfahren; der Vorfahre des einen Taxons wäre sozusagen der Bruder des Anderen.
Auf dem Weg von einem der beiden Taxa zum Anderen wechselt man am Ende zwangsweise in einen Merkmalszustand, den es bereits schon gab.
In Abbildung 10.5 lässt sich dieser Sachverhalt anschaulich betrachten.

Konvexität Eine Formulierung mit graphentheoretischen Begriffen führt zum Konvexitätsbegriff für elternbeschriftete Phylogenien.

Definition *Ein Merkmal ist konvex, wenn für jedes seiner Zustände höchstens ein Teilbaum in der Phylogenie existiert*

Wenn wir uns die oben beschriebene Situation ansehen und für einen bestimmten Zustand Teilbäume in der Phylogenie erstellen, werden wir einen Wald als Ergebnis haben. Haben zwei Taxa für ein Merkmal nicht den gleichen Vorfahren(Homoplasie), so liegt ein Vorfahre dazwischen, der das Merkmal nicht besitzt. Dieser Vorfahre wird ausgelassen, da der Teilbaum nur Taxa hinzunimmt, die das Merkmal besitzen. Dadurch wird klar, dass man mit den Kanten aus der Phylogenie keinen zusammenhängenden Teilbaum erzeugen kann, da auf dem Weg von einem Taxon zum Anderen ein Knoten(Pfad) dazwischen fehlen würde. Was sich in Abbildung 10.6 bemerkbar macht.

In einer perfekten Phylogenie sind alle Merkmale konvex. Die Konvexität muss für alle Merkmale gelten, damit wir eine homoplasiefreie Phylogenie haben. Die maximale Parsimonie wurde bereits durch einen Steinerbaum erreicht.
Diese beiden Kriterien müssen gelten, damit wir es mit einer perfekten Phylogenie zutun haben.

10.5 Kladistische Phylogenien

Der vorherige Ansatz war eine Lösung für elternbeschriftete Graphen. Kladistische Phylogenien sind jedoch gewurzelte Bäume ohne Elternknotenbeschrifung. Man kann jedoch auch auf direktem Wege kladistischen Phylogenien berechnen.

Konvexität für blattbeschriftete Phylogenien Letztendlich suchen wir kladistische, blattbeschriftete Darstellungen für Phylogenien. Insofern kann es von Nutzen sein den Konvexitätbegriff für blattbeschriftete Phylogenien umzuformulieren.

Definition *Minimale Teilbäume eines Merkmals sind für seine verschiedenen Zuständen disjunkt:*

$T_{m,\gamma}$ betrachte alle Knoten, die für Merkmal m den Zustand γ besitzen. Sei $T_{m,\gamma}$ ein zusammenhängender minimaler Teilbaum , dann soll gelten, $T_{m,\alpha} \cap T_{m,\beta} = \emptyset$

Nach dieser Definition nehmen wir alle Knoten mit, die die Taxa verbinden. Hypothetische Vorfahren existieren bei blattbeschrifteten Phylogenien nicht und gelten also nicht mehr als merkmalbesitzende Knoten. Knoten, die in elternbeschrifteten Bäumen ein Merkmal nicht besessen hätten, wären ausgelassen worden, *hier jedoch werden sie hinzugenommen* um einen minimalen Baum zu bilden. Klar sollte sein, dass ein Knoten der nach erster Konvexität ausgelassen worden wäre, als Verzweigung für Taxa dient, die einen anderen Zustand für das betrachtete Merkmal besitzen. Was wörtlich bedeuten, dass wir nun einen Knoten haben, der auch in einem anderen Teilbaum enthalen ist. In diesem Fall wären die minimalen Teilbäume eines Merkmals nicht disjunkt.

Hieraus sollte erkennbar sein, dass die Konvexität für blattbeschriftete Phylogenien auch der für Elternbeschriftete entspricht. Eine blattbeschriftete Phylogenie besitzt also auch immer eine elternbeschriftete Phylogenie.

10.5.1 Beschraenkung auf binaere Merkmale

Im Folgenden werden wir uns auf binäre Merkmalszustände beschränken, was einige algorithmische Vorteile mit sich bringen wird, die später noch einmal besprochen werden. Vorher wenden wir uns aber ein paar formalen Aspekten zu.

Definition *Eine Merkmalstaxamenge enthält alle Taxa, die ein bestimmtes Merkmal besitzen:*
Sei T die Taxamenge und $Z_m = \{0, 1\}$ die Zustandsmenge für Merkmal m. Des weiteren sei $\zeta_m : T \to Z_m$. Die Merkmalstaxamenge sei beschrieben durch $M_m = \{t \in T : \zeta_m(t) = 1\}$.

Satz 2. *Eine Menge von binären Merkmalen besitzt genau dann eine perfekte Phylogenie, wenn alle Merkmalstaxamengen paarweise entweder disjunkt oder eine die Teilmenge der anderen ist.*

Wir betrachten diesen Satz genauer in einer Fallunterscheidung:
In jedem Fall bilden wir aus den Merkmalstaxamengen einen Baum in dem auf Höhe der Wurzel o.B.d.A noch kein Merkmal entstanden ist. Die Kanten des Baums stellen eine Merkmalsentstehung dar. Die Wurzel ist damit also merkmalslos.

- M_a und M_b sind disjunkt
  M_a und M_b teilen sich also kein Taxon.
  In M_a besitzen alle Taxa Merkmal a.
  In M_b besitzt kein Taxon Merkmal a.
  Bildet man einen Baum, so liegen alle Taxa aus M_a verzweigend von der Wurzel aus auf einer Seite und alle Taxa aus M_b auf der Gegenseite.
  Nach kladistischer Konvexität ist dann $T_{a,1} \cap T_{a,0} = \emptyset$, woraus folgt, dass M_a konvex ist.

  Für M_b mit $T_{b,1}$ und $T_{b,0}$ funktioniert das Ganze analog.

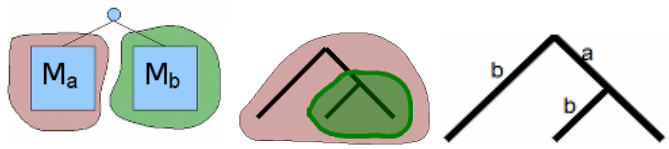


Abbildung 10.7: Links: M_a und M_b disjunkt. Mitte: $M_b \subseteq M_a$. Rechts: M_a und M_b weder Teilmengen von einander, noch disjunkt

- M_b ist Teilmenge von M_a

  Alle Taxa die Merkmal b besitzen, besitzen auch Merkmal a.
  In der Menge $M_a \cup M_b = M_a$ besitzen alle Taxa Merkmal a.
  Woraus folgt, dass $T_{a,0} = \emptyset$ ist.

  Nach kladistischer Konvexität folgt aus $T_{a,1} \cap T_{a,0} = T_{a,1} \cap \emptyset = \emptyset$, dass M_a konvex ist.

  Innerhalb M_b besitzen alle Taxa das Merkmal b.
  In M_a besitzt kein Taxon Merkmal b, da es ansonst bereits in der Menge M_b vertreten wäre.
  Damit ist in der Umgebung M_a /M_b $T_{b,0} = \emptyset$ also auch $T_{b,1} \cap T_{b,0} = \emptyset$ womit auch M_b konvex ist.

  Das einzige Problem das bestehen könnte wäre ein M_c , dass ebenfalls Teilmenge von M_a wäre und vielleicht ein Taxon mit Merkmal b besitzen könnte.
  Falls M_c Teilmenge von M_b ist, bleibt M_b konvex(s.o.).
  Falls M_c und M_b disjunkt sind ebenfalls(s.o.).
  Andernfalls ist M_b nicht konvex.

- M_a und M_b sind weder Teilmengen von einander, noch disjunkt.

  Es existiert also mindestens ein Taxon in M_a und M_b , dass sowohl Merkmal a als auch Merkmal b besitzt.
  Betrachten wir erst einmal M_a
  Verzweigt man in einen Teilbaum $T_{a,1} \cup T_{b,0}$, so muss später noch $T_{b,1}$ hinzukommen, da in M_a auch einige Taxa sind, die Merkmal b haben. Verzweigt man in einen Teilbaum $T_{a,1} \cup T_{b,1}$, so muss später noch $T_{b,0}$ hinzukommen.
  Für M_b gilt das Gleiche analog.

  In jedem Fall wäre die (Teil-)Wurzel der Verzweigung ein Knoten, der für $T_{b,1} \cap T_{b,0}$ nicht disjunkt sein kann, da er für eine Verbindung für einen minimalen Baum auf der Gegenverzweigung noch als Verbindungsknoten dienen müsste.

Diskussion Soweit haben wir die Berechnung perfekter Phylogenien durch Steinerbäume kennengelernt. Problem bei der Berechnung durch Steinerbäume ist ihre Laufzeit. Das Steinerbaumproblem ist NP-Vollständigen und kann nur durch Approximationen auf eine polynomielle Laufzeit gedrückt werden. Die Beschränkung auf binäre Merkmale hingegen lässt eine Laufzeit von O(tm) zu, für t Taxa und m Merkmale. Hinzu kommt noch, dass sich n-näre Zustandsalphabeta auf binäre faktorisieren lassen, wodurch sich das perfekte Phylogenienproblem für n-näre Zustände mit einer Laufzeit von O(nmt) berechnen lässt. Die Berechnung für Phylogenien bleibt hingegen auf perfekte Phylogenien beschränkt.

Literaturverzeichnis

CpG-Insel, 2009. URL `http://www.diss.fu-berlin.de/diss/servlets/MCRFileNodeServlet/FUDISS_derivate_000000000325/08_kap4.pdf`. [Online; Zugriff am 19.1.2009].

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis.Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998a.

R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998b.

D. Fernandez-Baca. The perfect phylogeny problem.

G. Fink. *Mustererkennung mit Markov-Modellen.Theorie - Praxis - Anwendungsgebiete*. B.G. Teubner Verlag, 2003.

A. J. Gibbs. The diagam, a method for comparing sequences. *European Journal of Biochemistry*, 1970.

O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982.

Krebsinformationsdienst, 2009. URL `http://www.krebsinformationsdienst.de/themen/grundlagen/index.php`. [Online; Zugriff am 20.12.2008].

H. Luz. *Online Lectures on Bioinformatics*. Max Planck Gesellschaft. http://lectures.molgen.mpg.de/Variants/SuboptAli/index.html.

J. V. Maizel. Enhanced graphic matrix analysis of nucleic acid and protein sequences. *Proceedings of the United States National Academy of Science*, 1981.

R. Merkl and S. Waack. *Bioinformatik Interaktiv.Algorithmen und Praxis*. WILEY-VCH Verlag, 2003a.

R. Merkl and S. Waack. *Bioinformatik interaktiv: Algorithmen und Praxis*. Wiley-VCH, 2003b.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

M. S. Waterman and M. Eggert. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *Journal of Molecular Biology*, 197:723–728, 1987.

Wikipedia. Substitutionsmatrix, 2009a. URL `http://de.wikipedia.org/wiki/Substitutionsmatrix`. [Online; Zugriff am 21.01.2009].

Wikipedia. Taxon. Wikipedia, The Free Encyclopedia, 2009b. URL `http://de.wikipedia.org/wiki/Taxon`. [Online; Zugriff am 20.03.2009].

Wikipedia: Krebs, 2009. URL `http://de.wikipedia.org/wiki/Krebs_(Medizin)`. [Online; Zugriff am 20.12.2008].

W. J. Wilbur. Rapid similiarity searches of nucleic acid protein data banks. *Proceedings of the United States National Academy of Science*, 1983.