Johannes Gutenberg-Universität
Mainz

Multiple Sequence Alignment - Komplexitätstheoretische Betrachtung und Vergleich von Verfahren

Diplomarbeit

vorgelegt dem Fachbereich Mathematik und Informatik
im Februar 2002
von

Götz Schwandtner
Finther Straße 19b
55257 Budenheim

Das Thema stellte
Herr Prof. C. Lautemann

Einleitung

Verschiedene Forschungsbereiche in der Biologie widmen sich der Untersuchung von DNA-Sequenzen. Diese stellen Bauplan von Lebewesen dar und enthalten die eine Codierung der Abfolgen der Aminosäuren in Proteinen.

Da die meisten Abläufe in Organismen von Proteinen gesteuert werden, sind sie die wichtigsten Makromoleküle in den Lebewesen. Die Informationen über den Aufbau der Proteine werden als DNA von Generation zu Generation weitervererbt, wobei Mutationen geschehen können. Mutationen in den für die Funktion wichtigen Bereichen der DNA führen oft dazu, dass Lebewesen mit dieser Mutation nicht mehr überlebensfähig sind und aussterben. Durch diese natürliche Selektion kommt es zu einer Erhaltung von Eigenschaften bei größeren Populationen, die sich als Konservierung von Bereichen der DNA auswirkt. Je nach Spezies enthalten nur wenige Bereiche der DNA wirklich verwendete Informationen, die sogenannten Exons, während der Rest (die Introns) keine solche Information über Proteine besitzen und deshalb Mutationen in diesen Bereichen meistens keine Auswirkungen haben und dementsprechend häufig zu beobachten sind.

Um Aussagen über die Verwandtschaft von Spezies anhand ihrer DNA- oder Proteinsequenzen treffen zu können, wird ein Verfahren benötigt um diese zu vergleichen. Dabei kann man keine Stringvergleiche einsetzen, da Mutationen zu unterschiedlich langen Sequenzen und auch veränderten Sequenzbereichen führen, ohne notwendigerweise die Funktion zu beeinträchtigen. Also wird das sogenannte paarweise Alignment eingesetzt, das Aussagen über die Ähnlichkeit von zwei Sequenzen liefert. Eine natürliche Erweiterung dieses Verfahren ist das Multiple Sequence Alignment, bei dem mehr als zwei Sequenzen gleichzeitig verglichen werden und damit Aussagen über Gemeinsamkeiten der Sequenzen und Verwandtschaften von mehreren Spezies erlaubt. Diese Verallgemeinerung führt jedoch zu einer hohen Komplexität des Verfahrens, so dass es im allgemeinen zu aufwendig ist, ein optimales Alignment zu bestimmen.

Ein verwandtes Problem ist das Tree-Alignment, bei dem es darum geht, zu einer Sequenzmenge einen phylogenetischen Baum zu finden, also einen Baum, der die Verwandtschaftsverhältnisse beschreibt. Da es sich dabei auch um ein Problem hoher Komplexität handelt, werden auch komplexitätstheoretische Aussagen dazu angegeben, ohne allerdings genau auf die Aspekte des Problems einzugehen.

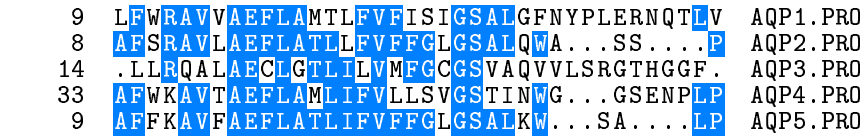


Abbildung 1: Beispiel für ein Multiple Sequence Alignment, gesetzt mit TEXshade aus [Bei00]

Nun zu Gliederung dieser Arbeit: Zunächst werden die benötigten Begriffe aus Bio-Informatik und Komplexitätstheorie eingeführt. Im Kapitel 2 folgt dann eine Komplexitäts-Analyse des Problems. Kapitel 3 umfasst einen empirischen Vergleich von Multiple-Sequence-Alignment-Verfahren, der mit einer kurzen Betrachtung zum Teil noch offener Fragen aus der Biologie in Kapitel 4 abgeschlossen wird.

Um die Lesbarkeit zu verbessern, sind die Literaturangaben jeweils am Ende des Kapitels in einem mit „Anmerkungen“ bezeichneten Abschnitt zu finden. Im Anhang befinden sich Informationen über Programme und die beigelegte CD.

Inhaltsverzeichnis

Kapitel 1

Definitionen

1.1 Multiple Alignment

1.1.1 Alignment von zwei Sequenzen

Das paarweise Alignment von zwei DNA- oder Aminosäuresequenzen dient zum Herausfinden von Ähnlichkeiten oder Unterschieden der Sequenzen. Damit lassen sich Verwandtschaftsbeziehungen der beiden Sequenzen oder das Vorkommen von Sequenzbruchstücken in längeren Sequenzen finden. Dazu werden die Sequenzen aneinander angeordnet und eventuell noch Lücken, sogenannte Gaps eingefügt. Formal läßt sich das wie folgt darstellen:

Definition 1.1 Paarweises Alignment

Sei Σ ein endliches Alphabet. Sei $\Delta \notin \Sigma$ das Zeichen für einen **Gap** *und $\tilde{\Sigma} := \Sigma \cup \{\Delta\}$*

Sei $s \in \Sigma^*$ eine **Sequenz**. *Dann bezeichne $s[i]$ das i-te Zeichen von s und $s[i..j] := s[i] \dots s[j]$ die Teilsequenz der Zeichen $s[i]$ bis $s[j]$.*

Für zwei Sequenzen $s_1, s_2 \in \Sigma^*$ ist ein **paarweises Alignment** *eine Abbildung*

$$\alpha : \{1, 2\} \to \tilde{\Sigma}^*$$

mit folgenden Eigenschaften:

1. $|\alpha(1)| = |\alpha(2)|$
2. $\alpha(i)\,|_\Sigma = s_i$ *für* $i \in \{1, 2\}$
3. *Für alle* $j \in \{1, \dots, |\alpha(1)|\}$ *gilt:* $\alpha(1)[j] \neq \Delta$ oder $\alpha(2)[j] \neq \Delta$

Ein paarweises Alignment ist also das Erweitern von zwei Sequenzen um Gaps und dann die zeichenweise Zuordnung der erweiterten Sequenzen, als $2 \times |\alpha(1)|$-Matrix (vergleiche z.B. Abbildung 1.1). Als Alphabet Σ werden meistens die Nukleobasen $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ oder die Aminosäuren verwendet. Ununterbrochene Folgen von Δ werden ebenfalls als *Gap* bezeichnet. Im folgenden

werden wir als Alignment das Bild der Abbildung α bezeichnen und folgende Abkürzung verwenden:

$$\alpha_{s_i} := \alpha(i) \tag{1.1}$$

Ein Alignment von zwei Sequenzen soll eine Aussage über die Ähnlichkeit der Sequenzen ermöglichen. Dazu benötigt man das nach bestimmten Kriterien bestpassende Alignment, um damit eine Aussage über die Ähnlichkeit der Sequenzen zu erhalten. Zunächst werden wir uns mit einem einfachen Kriterium beschäftigen, das aber leicht auf kompliziertere erweiterbar ist. Nach der Vorstellung des dynamischen Programmierens als Algorithmus zur Bestimmung eines optimalen Alignments werden dann einige dieser Erweiterungen angesprochen.

Definition 1.2 Match, Mismatch und Gap Penalty

Gegeben sei ein paarweises Alignment α der Sequenzen s_1 und s_2. Dann wird Spalte $(\alpha_{s_1}[i], \alpha_{s_2}[i])$ bezeichnet als

- **Match**, *falls* $\alpha_{s_1}[i] \in \Sigma$ *und* $\alpha_{s_1}[i] = \alpha_{s_2}[i]$,
- **Mismatch**, *falls* $\alpha_{s_1}[i], \alpha_{s_2}[i] \in \Sigma$ *und* $\alpha_{s_1}[i] \neq \alpha_{s_2}[i]$,
- **Insertion** *oder* **Deletion**, *falls* $\alpha_{s_1}[i] = \Delta$ *und* $\alpha_{s_2}[i] \in \Sigma$ *oder* $\alpha_{s_1}[i] \in \Sigma$ *und* $\alpha_{s_2}[i] = \Delta$.

Um die Qualität eines Alignments anhand dieser Begriffe zu beurteilen, wird jeder dieser drei Fälle mit einem Score versehen. Der (Gesamt-) Score des Alignments ist dann die Summe der Scores für alle Spalten und gibt ein Maß für den Abstand der beiden Sequenzen an. Mit $\mathrm{sc}(\sigma_1, \sigma_2)$ wird der Score für den Vergleich der beiden Zeichen σ_1, $\sigma_2 \in \tilde{\Sigma}$ angegeben. Mit $M := \mathrm{sc}(\sigma, \sigma)$ als Score für Match, $m := \mathrm{sc}(\sigma_1, \sigma_2)$ als Score für Mismatch ($\sigma_1 \neq \sigma_2$, $\sigma_1, \sigma_2 \in \Sigma$) und $g := \mathrm{sc}(\sigma, \text{-}) = \mathrm{sc}(\text{-}, \sigma)$, $\sigma \in \Sigma$ als Score für einen Gap ergibt sich dann beispielsweise [1]:

$M := 0, m := 1, g := 2$, $s =$ GACGGATTAG, $t =$ GATCGGAATAG

Alignment:	G	A	C	G	G	A	T	T	A	G	-	
	G	A	T	C	G	G	A	A	T	A	G	Gesamt:
Scores:	0	0	1	1	0	1	1	1	1	1	2	=9

Abbildung 1.1: Beispiel eines suboptimalen paarweisen Alignments

Das Alignment in Abb. 1.1 enthält viele Mismatches und ist offensichtlich nicht geeignet, die Verwandtschaftsbeziehungen zwischen den Sequenzen zu klären. Ein optimales Alignment für diese beiden Sequenzen und das angegebene Scoring-System ist in Abbildung 1.2 zu finden und zeigt deutlich die Ähnlichkeit der beiden Sequenzen.

[1] Üblicherweise wird bei der Darstellung von Alignments wegen der besseren Lesbarkeit ein - für einen Gap verwendet

Alignment:	G	A	-	C	G	G	A	T	T	A	G	
	G	A	T	C	G	G	A	A	T	A	G	Gesamt:
Scores:	0	0	2	1	0	0	0	1	0	0	0	=4

Abbildung 1.2: Beispiel eines optimalen paarweisen Alignments

Zum Bestimmen eines (im allgemeinen nicht eindeutig bestimmten) optimalen Alignments läßt sich die bekannte Methode des *dynamischen Programmierens* einsetzen. Dabei werden Teilergebnisse für Anfangsstücke berechnet und für die weitere Berechnung wiederverwendet. Im Fall des paarweisen Alignments werden die Scores der Teilsequenzen $s_1[1..i]$ und $s_2[1..j]$ sukzessive berechnet. Der Score eines optimalen Alignments von $s_1 = s_1[1..n]$ und $s_2 = s_2[1..m]$ ist dann der gesuchte Score eines optimalen Alignments der Sequenzen s_1 und s_2, wobei $n := |s_1|$ und $m := |s_2|$ ist.

Für den Algorithmus zur Bestimmung eines optimalen paarweisen Alignments benötigt man eine Matrix zum Speichern der Teilergebnisse:

Definition 1.3 Scorematrix a für das paarweise Alignment

Seien s_1 und s_2 Sequenzen über Σ mit Länge n bzw m wie oben, $1 \leq i \leq n$ und $1 \leq j \leq m$. Dann bezeichne $a[i,j] := \mathrm{sc}(s_1[1..i], s_2[1..j])$ den eindeutig bestimmten Score für ein optimales Alignment der Teilsequenzen $s_1[1..i]$ und $s_2[1..j]$.

Damit ist der Score für ein optimales Alignment von s_1 und s_2 in $a[n,m]$ zu finden. Wie schon erwähnt, kann diese Matrix unter Verwendung bereits berechneter Wert bestimmt werden:

$$a[0,0] := \quad 0 \tag{1.2}$$

$$a[i,0] := \quad i \cdot g \tag{1.3}$$

$$a[0,j] := \quad j \cdot g \tag{1.4}$$

$$a[i,j] := \quad \min\{ \quad a[i-1,j] + g, \quad a[i,j-1] + g, \quad a[i-1,j-1] + \mathrm{sc}(s_1[i], s_2[j]) \quad \} \tag{1.5}$$

$$\forall i \in \{1,\ldots,n\}, \quad j \in \{1,\ldots,m\}$$

Dabei ergibt sich Gleichung (1.5) aus folgender Überlegung: Für ein Alignment von $s_1[1..i]$ und $s_2[1..j]$ kann man um ein Zeichen kürzere Alignments berechnen und dann die Zeichen $s_1[i]$ bzw. $s_2[j]$ hinzufügen. Dabei gibt es drei Fälle, von denen einer mit dem optimalen Score verwendet wird: Im ersten werden die Präfixe $s_1[1..i-1]$ und $s_2[1..j]$ verwendet, weshalb dann $s_1[i]$ beim Hinzunehmen mit einem Δ zugeordnet werden muss. Analog funktioniert der Fall für $s_2[j]$. Im dritten Fall werden $s_1[i]$ und $s_2[j]$ zugeordnet und entsprechend an ein optimales Alignment von $s_1[1..i-1]$ und $s_2[1..j-1]$ angehängt.

Dann ergibt sich für die Scores $a[i,j]$:

$$\begin{aligned} a[i,j] &= \mathrm{sc}(s_1[1..i], s_2[1..j]) \\ &= \min\{ \quad \mathrm{sc}(s_1[1..i-1], s_2[1..j]) + \mathrm{sc}(s_1[i], \Delta), \\ &\qquad \mathrm{sc}(s_1[1..i], s_2[1..j-1]) + \mathrm{sc}(\Delta, s_2[j]), \\ &\qquad \mathrm{sc}(s_1[1..i-1], s_2[1..j-1]) + \mathrm{sc}(s_1[i], s_2[j]) \ \} \\ &= \min\{ \quad a[i-1,j] + g, \ \ a[i,j-1] + g, \ \ a[i-1,j-1] + \mathrm{sc}(s_1[i], s_2[j]) \ \} \end{aligned}$$

Die Berechnung der Matrix erfolgt zeilenweise von $a[0,0]$ bis $a[n,m]$, wobei alle bei der Berechnung von $a[i,j]$ verwendeten Zellen schon in vorherigen Schritten bestimmt wurden. Mit der Scorematrix a läßt sich natürlich nicht nur der Score eines optimalen Alignments bestimmen, sondern auch ein optimales Alignment selbst. Dazu werden einfach die Schritte der Berechnung von $a[n,m]$ zurück zum Anfang $a[0,0]$ verfolgt und entsprechend das Alignment von hinten nach vorne aufgebaut. Dabei gibt es manchmal mehrere Möglichkeiten, die beide zum gleichen Score führen. Um ein eindeutiges Alignment zu erhalten, wird eine Präferenz festgelegt, die dann bei gleichem Score die Fälle von (1.5) z.B. in der dort festgelegten Reihenfolge wählt.

Ein Beispiel für das Verfahren ist in Abb. 1.3 zu sehen.

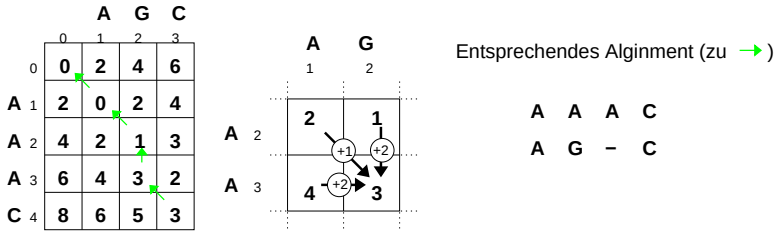


Abbildung 1.3: Beispiel für paarweises Alignment von `AAAC` und `AGC`

Manchmal wird die Score-Matrix mit den Pfeilen eines optimalen Alignments (wie in Abb. 1.3) auch als *Edit-Graph* bezeichnet. Damit läßt sich das Problem eines optimalen Alignments als graphentheoretisches Problem formulieren: Ein Edit-Graph ist ein gewichteter Graph $G = (V, E)$, dessen Knoten den Zellen der Scorematrix a entsprechen und dessen Kanten den Operationen: Anfügen von Match, Mismatch oder Gap zum bestehenden Teilalignment.

Es gibt also zu einem Knoten $v_{a[i,j]}$ genau die drei abgehenden Kanten (mit den dazugehörenden Gewichten $w(\cdot)$):

Kante e :	$w(e)$
$(v_{a[i,j]}, v_{a[i+1,j]})$	g
$(v_{a[i,j]}, v_{a[i+1,j+1]})$	$\mathrm{sc}(s_1[i+1], s_2[j+1])$
$(v_{a[i,j]}, v_{a[i,j+1]})$	g

Ein kürzester Weg von $v_{a[0,0]}$ bis $v_{a[n,m]}$ entspricht dann gerade einem optimalen Alignment.

Aus biologischer Sicht interessant sind auch andere Scoring-Systeme und Alignment-Algorithmen. Manche bestrafen Gaps am Anfang oder Ende von Sequenzen nicht, wieder andere haben einen festen Score für Gaps beliebiger Länge (*fixed Gap Penalty*) oder einen festen Score für Gaps mit einer Länge oberhalb einer vorgegebenen Schranke. Mit einer leicht abgewandelten Version des vorgestellten Verfahren lassen sich auch noch beliebige affin-lineare Gap-Penalties realisieren:

Definition 1.4 Affin-lineare Gap-Penalty-Funktionen

Für $h, b \geq 0$ wird $g(w) := h + bw$ als **affin-lineare Gap-Penalty-Funktion** *in Abhängigkeit von der Gap-Länge w bezeichnet.*

Ist $b = 0$, so liegt eine **fixed Gap Penalty-Funktion** *vor.*

Statt der Distanz kann man auch den Grad der Übereinstimmung von zwei Sequenzen als Score benutzen und diesen dann maximieren. Außerdem kann zwischen verschiedenen Matches und Mismatches unterschieden werden, um biologisches Wissen über Mutationen in die Bewertung aufzunehmen. Diese Scoring-Systeme lassen sich als Matrix formulieren, bei denen die Einträge die Scores für das Aufeinandertreffen von zwei Zeichen im Alignment angeben.

Beispielsweise kann man die Nukleobasen (`A`, `C`, `G` und `T`) aufgrund ihrer chemischen Struktur in zwei Gruppen einteilen: Purine (Adenin `A` und Guanin `G`) und Pyrimidine (Cytosin `C` und Thymin `T`). Dann wird eine Mutation innerhalb einer solchen Gruppe als Transition bezeichnet und eine zwischen den Gruppen als Transversion. Da die Nukleobasen einer Gruppe eine ähnliche chemische Struktur haben, ist z.B. eine Transition `A` $\rightarrow$ `G` wahrscheinlicher als eine Transversion `A` $\rightarrow$ `C`. In Abbildung 1.4 ist ein Beispiel für ein fiktives Scoringsystem zu sehen, bei dem das berücksichtigt wurde: Transitionen haben eine geringe Distanz, Transversionen eine höhere zur Folge.

	Δ	A	G	T	C
Δ	-	3	3	3	3
A	3	0	1	2	2
G	3	1	0	2	2
T	3	2	2	0	1
C	3	2	2	1	0

Abbildung 1.4: Beispiel eines Scoring-Systems für Nukleobasen

Definition 1.5 Scoring-System

Für ein endliches Alphabet Σ ist ein Scoring-System eine $(|\tilde{\Sigma}| \times |\tilde{\Sigma}|)$-Matrix, die die Werte der Funktion

$$\mathrm{sc}(\cdot,\cdot) : \tilde{\Sigma} \times \tilde{\Sigma} \rightarrow \mathbb{N}$$

angibt, mit der die Distanz zwischen zwei Zeichen aus $\tilde{\Sigma}$ beschrieben wird.

Im folgenden nehmen wir $\mathrm{sc}(\cdot,\cdot)$ immer als Distanzfunktion an, solange es nicht explizit anders erwähnt wird.

Von den Scoring-Systeme, die in der Praxis oft eingesetzt werden, werden in Abschnitt 3.3.2 in Kapitel 3 zwei Beispiele vorgestellt. Ein häufig anzutreffender Begriff in diesem Zusammenhang ist die Identität von zwei Sequenzen, die meistens als Anteil der Matches in einem optimalen Alignment definiert ist.

Die folgende Bedingung erscheint durchaus biologisch sinnvoll:

Definition 1.6 Dreiecks-Ungleichung für Scoringsysteme

Für ein Scoring-System $\mathrm{sc}(\cdot,\cdot)$ *gilt die Dreiecks-Ungleichung, falls für alle* $x, y, z \in \Sigma$ *gilt:*

$$\mathrm{sc}(x,z) \leq \mathrm{sc}(x,y) + \mathrm{sc}(y,z) \tag{1.6}$$

Die Dreiecks-Ungleichung (1.6) bedeutet einfach ausgedrückt, dass eine Mutation über einen Umweg (hier y) nicht einfacher oder schneller gehen kann als auf direktem Wege.

Eine weitere Variante des paarweisen Alignments ist das *lokale Alignment*, bei dem die Teilstrings der beiden Sequenzen gesucht werden, deren Alignment unter allen Alignments von Teilstrings den höchsten Score hat.

Für alle bisher angesprochenen Scoring-Systeme ist es mittels dynamischen Programmierens möglich, ein optimales Alignment von zwei Sequenzen in Laufzeit $\mathcal{O}(n^2)$ zu bestimmen, wobei n wie üblich die Eingabelänge ist[2]. Es gibt auch beschleunigte Varianten des Verfahrens für das Alignment von sehr ähnlichen Sequenzen und solche, die statt $\mathcal{O}(n^2)$ nur $\mathcal{O}(n)$ Platz benötigen, aber dafür eine wesentlich längere Laufzeit haben.

1.1.2 Multiple Alignment mit SPScore

Oft kann es sinnvoller sein, mehrere Sequenzen gleichzeitig zu alignen. Das ist eine einfache Erweiterung des paarweisen Sequenzvergleichs, doch komplexitätstheoretisch um einiges schwieriger als dieses.

Definition 1.7 Multiple Alignment

Seien $s_1, \ldots, s_k$ *Sequenzen über* Σ. *Ein Multiple Alignment* α *ist eine Abbildung*

$$\alpha : \{1, \ldots, k\} \to \tilde{\Sigma}^*$$

die folgende Eigenschaften erfüllt:

1. $|\alpha(1)| = |\alpha(i)| \quad \forall i \in \{1, \ldots, k\}$
2. $\alpha(i)\,|_\Sigma = s_i \quad \forall i \in \{1, \ldots, k\}$
3. *Für jedes* $j \in \{1, \ldots, |\alpha(1)|\}$ *gibt es ein* $i \in \{1, \ldots, k\}$ *mit:* $\alpha(i)[j] \neq \Delta$

[2] Für die meisten Problemstellungen ist n die Länge der längeren Sequenz.

Auch hier bietet sich wieder die Matrix-Schreibweise an. Dabei sichert die Bedingung *3* zu, dass es keine Spalten im Alignment gibt, die nur aus dem Gap-Zeichen - bestehen. Auch im Fall des Multiple Alignment wird die aus Gleichung (1.1) bekannte Kurzschreibweise verwendet:

$$\alpha_{s_i} := \alpha(i) \quad \forall i \in \{1, \ldots, k\}$$

In Abbildung 1.5 ist eine $k \times |\alpha_{s_1}|$-Matrix zur Darstellung eines Multiple Alignment mit $k = 4$ abgebildet.

```
M  Q  P  I  L  L  L
M  L  R  -  L  L  -
M  K  -  I  L  L  L
M  P  P  V  L  I  L
```

Abbildung 1.5: Beispiel eines Multiple Alignments der Sequenzen `MQPILLL`, `MLRLL`, `MKILLL` und `MPPVLIL`

Das Bewertungsschema ist ebenfalls eine Erweiterung des Scores beim paarweisen Alignment und wird *Sum-of-Pairs-Score* (SPScore) genannt. Der SPScore eines Alignments α wird wie beim paarweisen Alignment spaltenweise berechnet und dann zum Gesamtscore addiert. Dabei bezeichne $\alpha[j]$ die j-te Spalte von α, $\mathrm{sc}(\cdot,\cdot)$ sei eines der Scoring-Systeme für paarweises Alignment und zusätzlich gelte: $\mathrm{sc}(\Delta, \Delta) := 0$.

$$\mathrm{SPScore}(\alpha[j]) := \sum_{1 \le i,i' \le k} \mathrm{sc}(\alpha_{s_i}[j], \alpha_{s_{i'}}[j]) \tag{1.7}$$

$$\mathrm{SPScore}(\alpha) := \sum_{j=1}^{|\alpha_{s_1}|} \mathrm{SPScore}(\alpha[j]) \tag{1.8}$$

Die Bezeichnung SPScore kommt von der Berechnung des Scores für eine Spalte des Alignments als Summe über alle paarweisen Scores. Für Spalte 3 des Beispiels in Abb. 1.5 ergibt sich z.B.:

$$\mathrm{SPScore}\begin{pmatrix} \mathtt{P} \\ \mathtt{R} \\ - \\ \mathtt{P} \end{pmatrix} = \mathrm{sc}(\mathtt{P},\mathtt{R}) + \mathrm{sc}(\mathtt{P},-) + \mathrm{sc}(\mathtt{P},\mathtt{P}) + \mathrm{sc}(\mathtt{R},-) + \mathrm{sc}(\mathtt{R},\mathtt{P}) + \mathrm{sc}(-,\mathtt{P})$$

Vertauscht man in (1.7) und (1.8) die Summation über Zeilen und Spalten von α, so erhält man eine Aussage von Qualität des Alignments im Zusammenhang mit dem Score der *Projektionen* $(\alpha_{s_i}, \alpha_{s_{i'}})$:

$$\mathrm{SPScore}(\alpha) = \sum_{1 \le i < i' \le k} \mathrm{sc}(\alpha_{s_i}, \alpha_{s_{i'}}) \tag{1.9}$$

Zum Berechnen eines optimalen Multiple Alignments mit SPScore kann einfach die Erweiterung des dynamischen Programmierens für zwei Sequenzen verwendet werden. Dann ist die Matrix a k-dimensional und jede Dimension steht für eine der k Sequenzen.

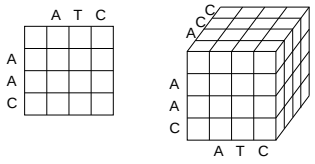


Abbildung 1.6: Matrix a für Sequenzen `AAC` und `ATC`, sowie diese beiden und zusätzlich `ACC`

Für k Sequenzen läßt sich die Verwendung schon berechneter Werte beim dynamischen Programmieren wie folgt darstellen:

Definition 1.8 Dynamisches Programmieren für k Sequenzen $s_1, \ldots, s_k$

Mit $\mathfrak{i} := (i_1, \ldots, i_k)$ *sei ein Multiindex bezeichnet und* $a[\mathfrak{i}] = a[i_1, \ldots, i_k]$ *sei der Score eines optimalen Alignments der Präfixe* $s_1[1..i_1], \ldots, s_k[1..i_k]$.

$\text{Column}(\mathfrak{i}, \mathfrak{b}) := (c_1, \ldots, c_k)$ *bezeichne die Spalte der bei dynamischen Programmieren in einem Schritt zum Alignment hinzugefügten Zeichen, wobei*

$$c_j := \begin{cases} s_j[i_j] & \text{falls} \quad b_j = 1 \\ \Delta & \text{falls} \quad b_j = 0 \end{cases}$$

und $\mathfrak{b} \in \{0,1\}^k$.

Hier gibt $\mathfrak{b}$ also die Herkunft der hinzugefügten Zeichen an: Bei $b_j = 1$ wird von Sequenz s_j ein Zeichen hinzugenommen, bei $b_j = 0$ stattdessen ein Gap.

Damit ergibt sich für die iterative Berechnung der Scorematrix:

$$a[\mathfrak{i}] = \max_{\substack{\mathfrak{b} \in \{0,1\}^k \\ \mathfrak{b} \neq (0,\ldots,0)}} \{a[\mathfrak{i} - \mathfrak{b}] + \text{SPScore}(\text{Column}(\mathfrak{i}, \mathfrak{b}))\} \tag{1.10}$$

Die eigentliche Berechnung wird auch hier wieder zeilenweise von $a[0, \ldots, 0]$ bis $a[|s_1|, \ldots, |s_k|]$ durchgeführt. Da der Aufwand für die Berechnung eines SPScore $\mathcal{O}(k + (k-1) + \cdots + 2 + 1) = \mathcal{O}(k^2)$ ist und die Berechnung jeder Zelle auf $2^k - 1 = \mathcal{O}(2^k)$ Vorgängerzellen zurückgreift, ergibt sich für die $\mathcal{O}(n^k)$ Zellen ein Gesamtaufwand von: $\mathcal{O}(k^2 2^k n^k)$.

Dieser Aufwand ist schon für kleine k groß und führt in der Praxis dazu, dass die exakte Lösung für mehr als 3 Sequenzen nicht mehr in annehmbarer Zeit bestimmt werden kann.

Im Abschnitt 3.4 von [SM97] wird eine beschleunigte Version des Verfahrens vorgestellt, die von Carillo und Lipman entwickelt wurde. Die dort zugrundeliegende Überlegung ist, dass bei der Berechnung eines optimalen Alignments nur

Zellen in einem Band um „die Hauptdiagonale“ [3] verwendet werden, was für das Alignment bedeutet, dass bei einem optimalen Alignment die Anzahl der Gaps nicht sehr groß ist.

Um nicht zur Berechnung eines optimalen Alignments benötigte Zellen auszuschließen, wird ein effizienter ($\mathcal{O}(n^2k^2)$) Relevanztest erstellt, der die Berechnung auf die relevanten Zellen beschränkt. Diese befinden sich in der Nähe der zu einem optimalen Alignment beitragenden Zellen. Dazu wird der Score einer Näherungslösung für ein optimales Alignment benötigt, wie sie z.B. mit dem Center Star Alignment (Definition 2.22) bestimmt werden kann. Je nach Güte dieser Näherung kann jedoch der Relevanztest versagen, so dass wieder alle Zellen verwendet werden. Dann ist mit $\mathcal{O}(k^4 2^k n^k)$ der Aufwand sogar größer als ohne Relevanztest. Das ist auch nicht weiter verwunderlich, da Multiple Alignment mit SPScore ein NP-vollständiges Problem ist, wie in Abschnitt 2.1.1 gezeigt wird. Das dynamische Programmieren mit Relevanztest wird in einer modifizierten Version im Programm MSA eingesetzt, das in Abschnitt 3.2.1 vorgestellt wird, wo auch eine genauere Beschreibung zu finden ist.

1.1.3 Tree Alignment

Für die Analyse genetischer Evolution kann man phylogenetische Bäume verwenden, welche wie folgt definiert sind:

Gegeben sei eine Menge S von k Sequenzen über dem Alphabet Σ. Sei S' eine weitere Menge von k' Sequenzen über dem gleichen Alphabet Σ. Weiter sei ein Scoringsystem $\mathrm{sc}(\cdot,\cdot)$ für paarweise Alignments gegeben. Mit $T_{(S,S')} = (E, V)$ wird dann ein ungerichteter, gewichteter Graph mit folgenden Eigenschaften bezeichnet:

1. $T_{(S,S')}$ ist ein Baum (nicht notwendigerweise mit Wurzel)
2. $T_{(S,S')}$ hat genau k Blätter v_s, wobei v_s mit jeweils einer verschiedenen Sequenz $s \in S$ markiert ist.
3. Falls $T_{(S,S')}$ eine Wurzel hat, so ist diese kein Blatt.
4. Die k' inneren Knoten von $T_{(S,S')}$ sind mit den verschiedenen Sequenzen aus S' markiert.
5. Für jede Kante $\{v_{s_1}, v_{s_2}\} \in E$ ($s_i \in S \cup S'$) sind die Kosten der Kante durch den Score des paarweisen Alignments von s_1 und s_2 definiert: $c(\{v_{s_1}, v_{s_2}\}) := \mathrm{sc}(s_1, s_2)$

$T_{(S,S')}$ ist dann ein *Tree Alignment* der Sequenzen S und die Kosten des Tree Alignments sind als Summe der Kantengewichte definiert:

$$\mathrm{sc}(T_{(S,S')}) := \sum_{\{v_1,v_2\}\in E} c(\{v_1, v_2\}) = \sum_{\{v_{s_1},v_{s_2}\}\in E} \mathrm{sc}(s_1, s_2) \tag{1.11}$$

[3] Bei $|s_1| = \ldots = |s_k| = n$ ist das die Diagonale $\{a[i, \ldots, i] \mid i = 1, \ldots, n\}$

Definition 1.9 Tree Alignment-Problem

Gegeben:	*Menge* $S = \{s_1, \ldots, s_k\}$ *von Sequenzen über einem endlichen Alphabet* Σ *und ein Scoringsystem* $\mathrm{sc}(\cdot, \cdot)$
Lösungen:	*Menge* S' *von Sequenzen über* Σ *und Baum* $T_{(S,S')}$
Bewertung:	$\mathrm{sc}(T_{(S,S')})$
Optimum:	*Minimum*

Die Sequenzen S entsprechen dabei lebenden Spezies, deren Verwandtschaft man untersuchen will und die Sequenzen in S' entsprechen potentiellen Vorfahren. Der Baum gibt dann die Verwandtschaftsverhältnisse an. Deshalb wird ein optimales Tree Alignment auch als *phylogenetischer Baum* bezeichnet.

Aber es gibt auch eine andere Variante des **Tree Alignment-Problems**, bei der biologisches Wissen über Verwandtschaft Verwendung findet. In diesem Fall ist der phylogenetische Baum schon vorgegeben und es werden nur noch die Sequenzen von Vorfahren-Spezies, also die Menge S' gesucht.

Zu einer gegebenen Sequenzmenge S sei mit $T_{(S,\cdot)}$ die Menge aller Bäume $T_{(S,S')}$ bezeichnet, die sich nur durch die Menge S' und die Markierung der inneren Knoten unterscheiden.

Definition 1.10 Tree Alignment-Problem mit vorgegebener Phylogenie

Gegeben:	*Sequenzmenge* S *über einem endlichen Alphabet* Σ*, ein Scoring-System* $\mathrm{sc}(\cdot, \cdot)$ *und eine Phylogenie* $T_{(S,\cdot)}$
Lösungen:	*Sequenzmenge* S' *und* $T_{(S,S')} \in T_{(S,\cdot)}$
Bewertung:	$\mathrm{sc}(T_{(S,S')})$
Optimum:	*Minimum*

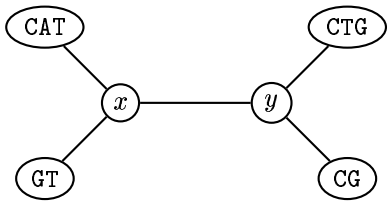


Abbildung 1.7: Beispiel für ein Tree-Alignment mit vorgegebener Phylogenie. $S = \{\mathtt{CAT}, \mathtt{CTG}, \mathtt{GT}, \mathtt{CG}\}$ und $S' = \{x, y\}$

Definition 1.11 Stern

Ein Baum T *mit Wurzel und Tiefe 1 wird als Stern bezeichnet.*

Bei diesem Graphen sind also alle Blätter direkt mit der Wurzel verbunden.

1.2 Komplexitätstheorie

Im folgenden Abschnitt werden die in dieser Arbeit verwendeten Komplexitätsklassen kurz definiert. Dabei werden jedoch bekannte Begriffe der Komplexitätstheorie wie Turing-Maschinen oder Sprachen als bekannt vorausgesetzt. Die Bezeichnungen von Entscheidungs- und Optimierungsproblemen sind in diesem und dem folgenden Kapitel stets in Fettdruck dargestellt, um sie von den zu Grunde liegenden biologischen Begriffen abzugrenzen.

1.2.1 P und NP

Die (Zeit-) Komplexitätsklasse **P** umfaßt alle Probleme die sich effizient lösen lassen, genauer: L bezeichne eine Sprache, M eine deterministische Turing-Maschine und L_M die Sprache, die von M entschieden wird. Dann ist **P** definiert als die Menge der Sprachen L für die eine polynomiell zeitbeschränkte Turingmaschine M existiert, derart dass $L = L_M$.

Die Anzahl der Berechnungsschritte ist also durch ein geeignetes Polynom in Abhängigkeit von der Eingabelänge n beschränkt und innerhalb dieser Zeit berechnet M zu jeder Eingabe-Instanz, ob es eine Lösung gibt oder nicht. Ein bekanntes Problem aus der Klasse **P** ist **2-SAT**, bei dem zu einer gegebenen booleschen Formel in konjunktiver Normalform mit 2 Literalen pro Klausel die Existenz einer erfüllenden Belegung gefragt wird. Dieses Problem ist eine einfache Variante des später betrachteten Problems **3-SAT**, das schwieriger ist. Algorithmen für Probleme in **P** können durch Polynome mit sehr hohem Grad oder sehr großen Koeffizienten beschränkt sein und deshalb nicht für die Berechnung mit großen Instanzen geeignet sein. Zwar liefert das „Speedup Theorem“ dann die Existenz eines schnelleren Algorithmus für die meisten Fälle, doch wird dann eine geänderte Codierung der Eingabe nötig.

Neben dieser Klasse der effizient lösbaren Probleme gibt es auch noch die Klasse **NP**, von der vermutet wird:

$$\mathbf{P} \neq \mathbf{NP} \tag{1.12}$$

Definition 1.12 Die Klasse **NP**

Eine Sprache L ist in **NP**, *falls es eine Turing-Maschine M gibt, so dass gilt:*

- $w \in L \Longleftrightarrow \exists y$, *so dass* $(w, y) \in L_M$
- *Die Laufzeit von M ist beschränkt durch ein Polynom in $|w|$*

Dabei wird mit L_M die Sprache bezeichnet, die von M akzeptiert wird und y als *Zusatzeingabe*. Diese Zusatzeingabe kann man auf zwei Arten deuten:

1. y ist ein Lösungskandidat der Instanz w und die deterministische Turingmaschine überprüft, ob diese Lösung gültig ist. Auf diese Weise kommt man ohne nichtdeterministische Turingmaschinen aus.

2. y bezeichnet einen akzeptierenden Berechnungsweg einer nichtdeterministischen Turingmaschine. In y ist also eine Entscheidung für alle nichtdeterministischen Stellen in der Berechnung der Turingmaschine codiert.

Offensichtlich sind alle Probleme in **P** auch in **NP** ($\mathbf{P} \subseteq \mathbf{NP}$) da man dann einfach die Zusatzeingabe ignorieren oder das leere Wort wählen kann: $y = \epsilon$. Unter der Voraussetzung (1.12) gibt es aber auch schwierige Probleme, die nicht in **P** liegen. Um Probleme nach Schwierigkeit ordnen zu können, bietet sich der Begriff der Reduktion an, mit der man Instanzen eines Problems mit einem Lösungsalgorithmus für ein anderes Problem lösen kann. Man kann also zeigen, dass ein Problem „mindestens so schwierig" wie ein anderes Problem ist.

Definition 1.13 polynomielle Reduktion $\leq_p$

Eine Sprache L ist auf eine Sprache L' **polynomiell reduzierbar** *($L \leq_p L'$), falls es eine in polynomieller Zeit berechenbare Funktion f gibt, so dass für alle w gilt:*

$$w \in L \Longleftrightarrow f(w) \in L'.$$

Diese Reduktion hat die wichtige Eigenschaft, dass die Klasse **P** abgeschlossen bzgl. $\leq_p$ ist, d.h. $L \in \mathbf{P}$, $L' \leq_p L \Rightarrow L' \in \mathbf{P}$. Damit ist man in der Lage, die potentiell schwierigen Probleme in **NP** zu identifizieren:

Definition 1.14 **NP**-hard und **NP**-vollständig

Eine Sprache L heißt **NP-hard**, *wenn für jede Sprache $L' \in \mathbf{NP}$ gilt: $L' \leq_p L$.*

Eine Sprache L heißt **NP-vollständig** *($L \in \mathbf{NPC}$), wenn L* **NP***-hard ist und $L \in \mathbf{NP}$.*

Um nachzuweisen, dass ein Problem L **NP**-hard ist, genügt es aufgrund der Transitivität von $\leq_p$, ein bereits bekanntes Problem $L' \in \mathbf{NPC}$ auf dieses zu reduzieren: $L' \leq_p L$. Denn dann können über L' alle anderen Probleme in **NP** auf L reduziert werden. Einer der bekanntesten Vertreter in **NPC** ist **3-SAT**, bei dem die Lösbarkeit eines booleschen Terms in konjunktiver Normalform mit drei Literalen je Klausel untersucht wird. Dass dieses Problem in **NP** ist, wird klar, indem man als Zusatzeingabe eine erfüllende Belegung des booleschen Terms als Lösungskandidat nimmt und dann diese für die Literale einsetzt und ausgewertet.

Falls für ein Problem in **NPC** gezeigt werden kann, dass es in **P** liegt, so folgt ein Kollaps der Klasse **NP** und es gilt $\mathbf{P} = \mathbf{NP}$ und damit nicht (1.12).

Da alle Laufzeitbetrachtungen asymptotische Untersuchungen sind, genügt es für den Nachweis, dass eine Sprache einer Klasse angehört, dies für alle Eingaben mit Länge $n \geq n_0$ zu betrachten, wobei n_0 eine geeignet gewählte Konstante ist. Das wird auch dadurch klar, dass man kleine Instanzen mit einer Länge bis zu n_0 zusammen mit ihrer Lösung in einer Tabelle ablegen kann, da es nur endlich viele sind. Für alle dort abgelegten Probleminstanzen kann in dieser Tabelle in polynomieller Zeit die Lösung ausgelesen und damit bestimmt werden.

1.2.2 Approximation

Neben den Entscheidungsproblemen gibt es auch noch Optimierungsprobleme, bei denen für jede Instanz eine möglichst gute Lösung gesucht ist. Die Qualität der Lösung läßt sich mittels einer in der Problemstellung gegebenen Kostenfunktion bestimmen und gesucht ist eine Lösung mit den minimalen Kosten im Falle eines Minimierungsproblems bzw. dem maximalen Gewinn im Falle eines Maximierungsproblems.

Definition 1.15 Optimierungsprobleme

Ein Problem A ist ein Optimierungsproblem, falls es zu jeder Instanz x eine Menge von Lösungen $F(x)$ gibt und eine Kostenfunktion $c : F(x) \to \mathbb{N}$, die jeder Lösung $s \in F(x)$ die Kosten $c(s)$ zuordnet.

Die optimalen Kosten $\mathrm{OPT}(x)$ *sind dann definiert als*

$\mathrm{OPT}(x) = \min_{s \in F(x)} c(s)$, *falls A Minimierungsproblem ist bzw.*

$\mathrm{OPT}(x) = \max_{s \in F(x)} c(s)$, *falls A Maximierungsproblem ist.*

Da die Berechnung einer Lösung mit optimalen Kosten, also eines Optimums, im allgemeinen zu aufwändig ist, versucht man mit Hilfe eines Approximationsverfahrens eine Näherungslösung zu bestimmen. Zur Charakterisierung der Qualität eines Verfahrens gibt es den Begriff des ε-Approximations-Algorithmus, der eine Aussage über den maximalen relativen Fehler erlaubt.

Definition 1.16 ε-Approximations-Algorithmus und relativer Fehler

Sei A ein Optimierungsproblem und M ein Algorithmus, der in polynomieller Zeit zu jeder Instanz x von A eine Lösung $M(x) \in F(x)$ bestimmt.

M heißt dann **ε-Approximations-Algorithmus**, *falls ein $\varepsilon > 0$ existiert, so dass für alle Instanzen x von A der relative Fehler $R(x, M)$ kleiner oder gleich ε ist:*

$$R(x, M) := \frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} \leq \varepsilon \tag{1.13}$$

In anderen Definitionen steht im Nenner $\mathrm{OPT}(x)$, was auf den ersten Blick offensichtlicher erscheint, doch die obige Definition ist für Maximierungs- und Minimierungsprobleme symmetrisch. ε nimmt nämlich für beide Arten von Problemen immer Werte zwischen 0 und 1 an. Mit einer einfachen Rechnung erhält man sogar eine Aussage über den Zusammenhang zwischen ε und den relativen Kosten $c(M(x))/\mathrm{OPT}(x)$:

Korollar 1.17 (Zusammenhang von ε mit den Ergebniskosten c)

1. *Ist A ein Maximierungsproblem, so gilt:* $\frac{c(M(x))}{\mathrm{OPT}(x)} \geq 1 - \varepsilon$
2. *Ist A ein Minimierungsproblem, so gilt:* $\frac{c(M(x))}{\mathrm{OPT}(x)} \leq \frac{1}{1-\varepsilon}$

Da der Wert ε vom Algorithmus abhängt, mit dem ein Problem gelöst werden soll, eignet er sich nicht für Nichtapproximierbarkeitsaussagen über Optimierungsprobleme. Dafür gibt es ähnlich wie bei der Definition von **P** und **NP** einen vom konkreten Algorithmus unabhängigen Begriff:

Definition 1.18 Approximations-Schranke

Als **Approximations-Schranke** *zu einem Optimierungsproblem A wird die größte untere Schranke aller $\varepsilon > 0$ bezeichnet, für die ein* **polynomiell zeitbeschränkter** *ε-Approximierungs-Algorithmus für A existiert.*

Falls die Ungleichung (1.12) gilt, können verschiedene **NP**-vollständige Probleme unterschiedlich große Approximationsschranken haben. Falls andererseits **P** = **NP** ist, haben offensichtlich alle Optimierungsprobleme in **NP** die Approximations-Schranke 0.

Neben dieser negativen Aussage gibt es auch einen Begriff, der beliebig genaue Approximierbarkeit in polynomieller Zeit zusichert:

Definition 1.19 Polynomial-Zeit-Approximations-Schema (**PTAS**)

Sei A ein Optimierungsproblem. Ein **PTAS** *ist dann ein laufzeitbeschränkter Algorithmus M, der zu jedem $\varepsilon > 0$ und jeder Instanz x von A eine Lösung $M_\varepsilon(x)$ liefert, so dass für den relativen Fehler gilt:*

$$R(x, M_\varepsilon) \leq \varepsilon \tag{1.14}$$

Dabei ist die Laufzeit für jedes ε durch ein Polynom in $|x|$ beschränkt.

Dieser Begriff hängt eng mit dem Begriff der Approximations-Schranke zusammen. So gibt es offensichtlich genau dann ein **PTAS** für ein Optimierungs-Problem, wenn dessen Approximations-Schranke 0 ist. Es gibt auch Probleme, deren Approximations-Schranke entweder 0 oder 1 ist. Findet man also einen ε-Approximations-Algorithmus für ein solches Problem mit $\varepsilon < 1$, so gibt es ein **PTAS** für dieses. Ein bekannter Vertreter dieser Klasse ist:

Definition 1.20 INDEPENDENT SET

Gegeben:	*Ungerichteter, einfacher Graph $G = (V, E)$*
Lösungen:	*Unabhängige Menge $I \subseteq V$, also:* *Für alle $v_1, v_2 \in I$ gilt $(v_1, v_2) \notin E$.*
Bewertung:	$c(I) := \|I\|$
Optimum:	*Maximum*

Falls **P**$\neq$ **NP**, so hat dieses Problem Approximations-Schranke 1. Um dies nachzuweisen, benötigt man eine Klasse der schwer approximierbaren Probleme, der **INDEPENDENT SET** angehört und die im folgenden definiert wird.

1.2.3 L-Reduktionen und MAXSNP

Analog zum Begriff der **NP**-Vollständigkeit und der Polynomialzeit-Reduktion $\leq_p$ für die Entscheidungsprobleme gibt es auch für Optimierungsprobleme einen Reduktionsbegriff und eine Klasse von schwierigen Problemen. Im Gegensatz zur Reduktion $\leq_p$ muss aber hier die Erhaltung der Approximierbarkeit sichergestellt werden.

Definition 1.21 L-Reduktion

Seien A und B zwei Optimierungsprobleme. Mit $I(A)$ und $I(B)$ seien alle Instanzen von A bzw. B bezeichnet, mit $F(x)$ die Lösungen einer Instanz x und mit $c_A(\cdot)$ und $c_B(\cdot)$ die Bewertungsfunktionen der beiden Probleme.

Zwei Funktionen R und S heißen dann eine **L-Reduktion** *von A nach B, kurz $A \leq_L B$, falls Konstanten β und γ existieren, so dass gilt:*

1. $R : I(A) \to I(B), x \mapsto R(x)$

   ist in logarithmischem Platz berechenbar

2. $S : F(R(I(A))) \to F(I(A)), s \mapsto S(s)$

   ist in polynomieller Zeit berechenbar und S liefert zu jeder Instanz $x \in I(A)$ und jeder Lösung $s \in F(R(x))$ eine Lösung von x:

   $$S : F(R(x)) \to F(x)$$

3. *Für jede Instanz $x \in I(A)$ gilt:*

   $$\mathrm{OPT}(R(x)) \leq \beta \mathrm{OPT}(x) \tag{1.15}$$

4. *Für jede Instanz $x \in I(A)$ und jede Lösung $s \in F(R(x))$ gilt:*

   $$|\mathrm{OPT}(x) - c_A(S(s))| \leq \gamma |\mathrm{OPT}(R(x)) - c_B(s)| \tag{1.16}$$

Mit L-Reduktionen läßt sich ein Zusammenhang zwischen der Approximierbarkeit verschiedener Probleme herstellen, da L-Reduktionen die Approximierbarkeit erhalten:

Satz 1.22 (L-Reduktionen sind approximierbarkeitserhaltend)

Seien A und B zwei Optimierungsprobleme und (R, S) eine L-Reduktion von A nach B mit den Parametern β und γ. Für B existiere ein ε-Approximations-Algorithmus. Dann existiert ein $\frac{\beta\gamma\varepsilon}{1-\varepsilon}$-Approximations-Algorithmus für A.

Korollar 1.23 (L-Reduktionen und **PTAS**)

Falls es eine L-Reduktion $A \leq_L B$ gibt und ein **PTAS** *für B, so gibt es auch ein* **PTAS** *für A.*

Um mit L-Reduktionen Komplexitätsklassen und vollständige Probleme definieren zu können, benötigt man noch folgende von anderen Reduktionen bekannte Eigenschaft:

Satz 1.24 (Hintereinanderausführung von L-Reduktionen)
Seien A, B und C Optimierungsprobleme, (R, S) eine L-Reduktion von A nach B mit den Parametern β und γ und (R', S') eine L-Reduktion von B nach C mit den Parametern β' und γ'. Dann ist $(R \circ R', S \circ S')$ eine L-Reduktion von A nach C mit den Parametern $\beta^* := \beta\beta'$ und $\gamma^* := \gamma\gamma'$, kurz:

$$A \leq_L B \text{ und } B \leq_L C \quad \Longrightarrow \quad A \leq_L C \tag{1.17}$$

Diese Eigenschaft werden wir bei der Definition der Klasse **MAXSNP** verwenden. Die folgende Definition der Klasse $\mathbf{MAXSNP_0}$ verwendet Begriffe aus der mathematischen Logik erster und zweiter Stufe, auf deren Definition hier verzichtet wird. Die auf $\mathbf{MAXSNP_0}$ aufbauende Definition von **MAXSNP** ist aber auch ohne Vorwissen aus diesem Bereich möglich.

Als Ausgangspunkt verwenden wir eine logische Charakterisierung von **NP**:

Satz 1.25 *(Satz von Fagin)*
Die Klasse aller in existenzieller Logik zweiter Stufe ausdrückbaren graphentheoretischen Eigenschaften ist genau die Klasse **NP**

Eine Teilmenge von **NP** ist die sogenannte Klasse *strict* **NP** bzw. **SNP**, die genau aus allen Eigenschaften besteht, die durch Aussagen der Form

$$\exists S \forall x_1 \forall x_2 \ldots \forall x_k \phi(S, G, x_1, \ldots, x_k) \tag{1.18}$$

beschreibbar sind, wobei ϕ eine quantorenfreie Formel der Logik erster Stufe ist, in der die Relation S, die Eingabe G und als freie Variablen $\{x_1, \ldots, x_k\}$ vorkommen. In (1.18) wird also beschrieben, dass für jede Eingabe $G \in$ **SNP** eine Relation S existiert, so dass für alle Tupel $(x_1, \ldots, x_k)$ die Formel ϕ erfüllt ist. Um von dieser Klasse von Entscheidungsproblemen zu einer Klasse von Optimierungsproblemen zu kommen, betrachten wir eine Abwandlung dieser Fragestellung, bei der eine Relation S gesucht ist, so dass die Menge der ϕ erfüllenden Tupel am größten wird.

Definition 1.26 $\mathbf{MAXSNP_0}$
Ein graphentheoretisches Maximierungsgsroblem A ist in $\mathbf{MAXSNP_0}$*, falls es sich wie folgt beschreiben läßt:*

$$\max_S |\{(x_1, \ldots, x_k) \in V^k \mid \phi(G_1, \ldots, G_m, S, x_1, \ldots, x_k)\}| \tag{1.19}$$

Dabei besteht die Eingabe zum Problem A aus den Relationen $G_1, \ldots, G_m$ über einem endlichen Universum V.

Ein Problem in $\mathbf{MAXSNP_0}$ ist **MAX-CUT**:

Definition 1.27 MAX-CUT

Gegeben: *Ungerichteter Graph* $G = (V, E)$

Lösungen: *Partition* $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$

Bewertung: $c(V_1) := |\{(v_1, v_2) \in E \mid v_1 \in V_1 \wedge v_2 \in V_2\}|$

Optimum: *Maximum*

Lemma 1.28 MAX-CUT *ist in* $\mathbf{MAXSNP_0}$

Als Beweis genügt folgende Darstellung von **MAX-CUT**:

$$\max_{S \subseteq V} |\{(x, y) \mid ((G(x, y) \vee G(y, x)) \wedge S(x) \wedge \neg S(y))\}|$$

Dabei gibt die Eingabe $G \subseteq V \times V$ die Kantenrelation des Graphen an und S entspricht V_1 aus Definition 1.27. □

In der Klasse $\mathbf{MAXSNP_0}$ sind nur graphentheoretische Optimierungsprobleme und auch nur Maximierungsprobleme zu finden. Aus diesem Grund betrachten wir folgende Erweiterung:

Definition 1.29 MAXSNP

Ein Optimierungsproblem A ist in **MAXSNP**, *falls es ein* $B \in \mathbf{MAXSNP_0}$ *gibt mit* $A \leq_L B$

Damit haben wir jetzt eine Klasse von Optimierungsproblemen, die auch Minimierungsprobleme und Probleme aus anderen Bereichen als der Graphentheorie enthält.

Mit dem Begriff der L-Reduktion lassen sich die schwierigen Probleme dieser Klasse bzw. bezüglich dieser Klasse identifizieren:

Definition 1.30 MAXSNP-hard und **MAXSNP**-vollständig

Ein Optimierungsproblem A heißt **MAXSNP-hard**, *falls für alle Probleme* $B \in \mathbf{MAXSNP}$ *gilt:* $B \leq_L A$.

Ist ein Problem sowohl in **MAXSNP** *als auch* **MAXSNP-hard**, *so wird es als* **MAXSNP-vollständig** *bezeichnet, kurz:*

$$\mathbf{MAXSNPC} := \mathbf{MAXSNP} \cap \mathbf{MAXSNP\text{-}hard}$$

Ein bekannter Vertreter dieser Klasse ist **MAX-CUT**:

Lemma 1.31 MAX-CUT *ist* **MAXSNP**-*vollständig*

Wegen der Konstruktion mit L-Reduktionen führt das Finden eines **PTAS** für ein Problem in **MAXSNPC** zu einem Kollaps von **MAXSNP**: Dann gibt es nämlich ein **PTAS** für alle Probleme in **MAXSNP**.

Als abschließende Bemerkung dieses Abschnitts ist ein Zusammenhang der Klasse $\mathbf{MAXSNP_0}$ und dem vorher betrachteten Begriffs des ε-Approximations-Algorithmus zu erwähnen:

Satz 1.32 Zusammenhang: ε-Approximations-Algorithmus und $\mathbf{MAXSNP_0}$

Sei A ein Problem in $\mathbf{MAXSNP_0}$ und habe (o.B.d.A.) die Gestalt $\max_S |\{(x_1, \ldots, x_k) \mid \phi\}|$. k_ϕ bezeichne die Anzahl der atomaren Ausdrücke von ϕ, in denen S vorkommt. Dann hat A einen $(1 - 2^{-k_\phi})$-Approximations-Algorithmus.

1.3 Anmerkungen

Die kurze Einführung in die Begriffe, die aus der Bio-Informatik stammen, ist an das Kapitel 3 von [SM97] angelehnt, wobei die Bezeichnungen für Alignments, SPScore und die Problemdefinitionen dem theoretischen Charakter dieser Arbeit angepaßt wurden. Das dynamische Programmieren zur Bestimmung eines optimalen Alignments von zwei Sequenzen ist auch als „Needleman-Wunsch-Algorithmus“ bekannt und wurde in [NW70] vorgestellt. Der Begriff des Edit-Graphen ist bei [Gus97] genauer definiert, während Transition, Transversion und deren Wahrscheinlichkeiten im Kapitel 3 von [LG00] genauer betrachtet werden. Die angesprochenen Varianten des paarweisen Alignments etwa mit geringerem Speicherbedarf sind in [SM97] genau erklärt.

Der Abschnitt 1.1.3 über Tree Alignment ist an [JL94] angelehnt.

Die Einführung in die verwendeten Begriffe der Komplexitätstheorie ist an zwei Büchern orientiert, die sich sehr gut als Einführung in dieses Gebiet eignen: [GJ79] und [Pap94]. Für eine Definition des dort verwendeten „Speedup Theorems“ vergleiche 7.4.13 in [Pap94]. Die Definition des Begriffs **NP** ist angelehnt an Definition 9.5 von [Sch99].

Die Abschnitte über Approximation und Optimierungsprobleme, 1.2.2 und 1.2.3, sind eng an das Kapitel 13 von [Pap94] angelehnt. Dort sind auch die Beweise der hier nicht bewiesenen Sätze, Lemmata und Aussagen zu finden. Satz 1.25 ist in [Pap94] als Theorem 8.3 aufgeführt und bewiesen.

Kapitel 2

Komplexitätstheoretische Betrachtung

2.1 NP-Vollständigkeit

Obwohl es sich bei den verschiedenen Varianten des Multiple Alignment um Optimierungsprobleme handelt, sind auch die Entscheidungsproblem-Versionen unter komplexitätstheoretischen Gesichtspunkten interessant.

2.1.1 Multiple Alignment ist NP-vollständig

Um die **NP**-Vollständigkeit eines Entscheidungsproblems nachzuweisen, geht man von einem als **NP**-vollständig bekanntem Problem aus und konstruiert eine Polynomialzeitreduktion von diesem auf das zu untersuchende Problem. (Vergleiche dazu auch Definition 1.14)

Für den Beweis, dass verschiedene Versionen von Multiple Alignment mit SPScore **NP**-vollständig sind, verwenden wir als Ausgangspunkt für die Reduktion folgendes Problem:

Definition 2.1 SIMPLE MAX-CUT-B[1]

Gegeben: *Einfacher, ungerichteter Graph* $G = (V, E)$ *mit* $1 \leq \deg(v) \leq B \ \ \forall v \in V$, $K > 0$

Frage: *Gibt es eine Partition* $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ *mit* $c(V_1) := |\{(v_1, v_2) \in E \mid v_1 \in V_1 \wedge v_2 \in V_2\}| \geq K$ *?*

Dieses Problem ist ein Spezialfall von MAX-CUT und für $B \geq 3$ **NP**-vollständig.

[1] Hier gibt es eine zusätzliche Annahme, die in den meisten Definitionen von **SIMPLE MAX-CUT-B** nicht vorkommt: $1 \leq \deg(v)$. Diese ändert jedoch nichts an der **NP**-Vollständigkeit von **SIMPLE MAX-CUT-B**, da die Reduktionen zum Nachweis davon ohne isolierte Knoten auskommen.

Für die Reduktion genügt es, Scoring-Systeme zu betrachten, die bestimmte Bedingungen erfüllen. Dabei gibt das Scoring-System hier Bestrafungen für Mismatches und Gaps an, d.h. größere Scores deuten auf ein schlechteres Alignment hin. Für die Reduktion wird zunächst ein Scoring-System für Alignments über dem Alphabet $\Sigma = \{\mathtt{A}, \mathtt{T}\}$ betrachtet, das wie in Kapitel 1 als Matrix dargestellt wird, wobei Δ wieder das Zeichen für einen Gap sein soll:

	Δ	A	T
Δ	0	y	z
A	y	$v_{\mathtt{A}}$	u
T	z	u	$v_{\mathtt{T}}$

Abbildung 2.1: Eine generische Score-Matrix

Definition 2.2 *Eine 3×3-Matrix mit den Bezeichnungen wie in Abb. 2.1 wird* **generisch** *genannt, falls $y, z \geq 0$ und $u > \max\{0, v_{\mathtt{A}}, v_{\mathtt{T}}\}$ gilt.*

Eine $(w+1) \times (w+1)$ Scoringmatrix N **enthält eine generische Untermatrix**, *falls die Zeilen und Spalten zu Δ und zwei Zeichen eine generische Matrix darstellen.*

Die Menge aller Scoringmatrizen, die eine generische Untermatrix enthalten, wird mit $\mathcal{M}_2$ bezeichnet, $\mathcal{M}_1 \subset \mathcal{M}_2$ bezeichnet die Scoringmatrizen, bei denen zusätzlich noch $z > v_{\mathtt{T}}$ gilt und $\mathcal{M} \subset \mathcal{M}_1$ diejenigen mit $y > u$.

Die Zeichen, die zu der generischen Teilmatrix gehören werden o. B. d. A. mit A und T bezeichnet. Wegen $y > u$ und $u > v_{\mathtt{A}}$ gilt für Scoring-Systeme in $\mathcal{M}$ nicht nur $z > v_{\mathtt{T}}$, sondern auch $y > v_{\mathtt{A}}$. Also werden alle Gaps stärker als die Matches der dazugehörigen Zeichen bestraft.

$\mathcal{M}_2$ umfaßt also die Scoring-Systeme, bei denen es ein Zeichen (in diesem Fall A) gibt, bei dem Mismatch (mit einem anderen Zeichen, hier T) gegenüber einem Match bestraft wird. $\mathcal{M}_1$ enthält noch eine Forderung an die Gap-Penalty, mit der das Einfügen eines Gaps Δ bestraft wird und in $\mathcal{M}$ soll eine Art Gap sogar stärker bestraft werden als ein Mismatch. Diese Einschränkungen sind für in der Anwendung benutzten Scoring-Systeme keine starken Einschränkungen und sogar die am meisten einschränkende Menge $\mathcal{M}$ enthält die meisten verwendeten Scoring-Systeme. Nicht berücksichtigt sind hier allerdings Scoring-Systeme mit fester Gap-Penalty oder konstanter Gap-Penalty für Gaps über einer bestimmten Länge.

Bei der folgenden Definition der Probleme ist das Scoring-System jeweils Teil der Eingabe. Der Beweis der **NP**-Vollständigkeit funktioniert aber genauso, wenn das Scoring-System nicht Teil der Eingabe ist, sich aber in der jeweils angegebenen Klasse $\mathcal{M}$, $\mathcal{M}_1$ bzw. $\mathcal{M}_2$ befindet. Um eine einheitliche Definition von Entscheidungsproblemen zu gewährleisten, wird hier ohne Einschränkung das Scoring-System als Teil der Eingabe angesehen.

Definition 2.3 MSA-SP

Gegeben: *Sequenzen* $\{s_1, \ldots, s_k\}$ *über einem endlichen Alphabet* Σ *und ein Scoring-System aus der Klasse* $\mathcal{M}$, $K > 0$

Frage: *Gibt es ein Multiple Alignment* α *der Sequenzen* $\{s_1, \ldots, s_k\}$ *mit* $\mathrm{SPScore}(\alpha) \leq K$

Definition 2.4 gap-0-MSA-SP

Gegeben: *Sequenzen* $\{s_1, \ldots, s_k\}$ *über einem endlichen Alphabet* Σ *und ein Scoring-System aus der Klasse* $\mathcal{M}_1$, $K > 0$

Frage: *Gibt es ein Multiple Alignment* α *der Sequenzen* $\{s_1, \ldots, s_k\}$ *mit* $\mathrm{SPScore}(\alpha) \leq K$, *wobei* α *keine internen Gaps enthält*

Definition 2.5 gap-0-1-MSA-SP

Gegeben: *Sequenzen* $\{s_1, \ldots, s_k\}$ *der gleichen Länge* $|s_1|$ *über einem endlichen Alphabet* Σ *und ein Scoring-System aus der Klasse* $\mathcal{M}_2$, $K > 0$

Frage: *Gibt es ein Multiple Alignment der Sequenzen* $\{s_1, \ldots, s_k\}$ *mit* $\mathrm{SPScore}(\alpha) \leq K$, *wobei für alle* $1 \leq i \leq k$ *gilt:*

- $|\alpha_{s_i}| = n := |s_1| + 1$
- *Entweder* $\alpha_{s_i}[1] = \Delta$ *oder* $\alpha_{s_i}[n] = \Delta$.

Eine **gap-0-1-MSA-SP**-Instanz enthält also insbesondere keine internen Gaps und in jeder Sequenz ist entweder ein Gap vor dem Anfang eingefügt oder am Ende angehängt. Die ursprünglichen Sequenzen $\{s_1, \ldots, s_k\}$ sind also einfach um höchstens ein Zeichen gegeneinander verschoben. Darauf baut auch die einfachste Version des **NP**-Vollständigkeits-Nachweises auf, mit dem der Beweis des folgenden Satzes anfängt:

Satz 2.6 (Multiple Alignment ist **NP−hard**)
Folgende oben definierte Varianten des Multiple Alignment sind **NP-hard***:*

1. **MSA-SP**
2. **gap-0-MSA-SP**
3. **gap-0-1-MSA-SP**

In dieser Form schließt Satz 2.6 zwar keine Scoring-Systeme ein, die Gaps am Anfang oder Ende der Sequenzen nicht bestrafen, aber im Beweis kann man leicht sehen, dass dies keine Einschränkung darstellt und der Satz auch für solche Alignments gilt.

Nun zu einem Überblick über den Beweis von 2.6:

Für den Beweis sei ein Wert für B gegeben, so dass **SIMPLE MAX-CUT-B** **NP**-vollständig ist. Im folgenden geben wir eine Reduktion von **SIMPLE**

MAX-CUT-B auf alle drei Varianten **MSA-SP**, **gap-0-MSA-SP** und **gap-0-1-MSA-SP** an.

Gegeben ist ein einfacher, ungerichteter Graph $G = (V, E)$ als **SIMPLE MAX-CUT-B**-Instanz. Seien $k := |V|$ die Anzahl der Knoten und $l := |E|$ die Anzahl der Kanten von G und seien die Elemente von G beliebig numeriert: $V = \{v_0, \ldots, v_{k-1}\}$, $E = \{e_0, \ldots, e_{l-1}\}$.

G wird wie folgt in k^2 Sequenzen $\bar{t}^G = \{t_0, \ldots, t_{k^2-1}\}$ der Länge $k^{12}l$ codiert: Jedem Punkt v_i entspricht die Sequenz t_i, während die restlichen Sequenzen $\{t_k, \ldots, t_{k^2-1}\}$ zusätzliche Zeilen sind, um die Score-Unterschiede zwischen Match und Mismatch zu vergrößern. Die Sequenzen bestehen überwiegend aus dem Zeichen `T`, das Zeichen `A` wird nur zur Codierung der Kanten verwendet: Für eine beliebige Kante $e_m = (v_h, v_i) \in E, v_h, v_i \in V, h < i$ setze für alle $n < k^5$:

$$t_{h,k^7ln+k^7m+2} = t_{i,k^7ln+k^7m+1} = t_{i,k^7ln+k^7m+3} = \texttt{A}$$

Ein Beispiel ist in Abbildung 2.2 zu finden.

		t_{g,k^7ln+k^7m} ↓		$t_{g,k^7ln+k^7m'}$ ↓		$t_{g,k^7ln'+k^7m}$ ↓	
t_g:	$\cdots$	T T T T T	$\cdots$	T T A T T	$\cdots$	T T T T T	$\cdots$
t_h:	$\cdots$	T T A T T	$\cdots$	T A T A T	$\cdots$	T T A T T	$\cdots$
t_i:	$\cdots$	T A T A T	$\cdots$	T T T T T	$\cdots$	T A T A T	$\cdots$
t_p:	$\cdots$	T T T T T	$\cdots$	T T T T T	$\cdots$	T T T T T	$\cdots$

Abbildung 2.2: Codierung von zwei Kanten $e_m = (v_h, v_i)$ und $e_{m'} = (v_g, v_h)$ des Graphen im Beweis von Satz 2.6. Für die Codierung der Kante e_m sind zwei Bereiche dargestellt, mit $n, n' < k^5$; für die Codierung der Kante $e_{m'}$ nur einer.

Diese Codierung wird für die Reduktion im Beweis von allen drei Aussagen von Satz 2.6 verwendet, doch um sie anschaulich zu erklären, betrachten wir hier den Fall des **gap-0-1-MSA-SP**. Bei einem **gap-0-1-MSA-SP** α von $\bar{t}^G$ kann entweder $\alpha_{t_i}[1] = \Delta$ oder $\alpha_{t_i}[k^{12}l] = \Delta$ gelten. Durch Setzen von

$$V_0^\alpha := \{v_i \in V : \alpha_{t_i}[1] \neq \Delta\} \quad \text{und} \quad V_1^\alpha := V \setminus V_0^\alpha$$

erhält man eine Partition von V. Die **gap-0-1-MSA-SP**-Instanz (also das Alignment mit minimalem Score) erzeugt gerade einen MAX-CUT (V_0^α, V_1^α) von G:

Seien α_{t_h} und α_{t_i} Sequenzen im **gap-0-1-MSA-SP** in denen eine Kante $e_m = (v_h, v_i)$ codiert ist. Dann gibt es zwei Fälle:

- $\alpha_{t_h} = \alpha_{t_i}$

  Das bedeutet, dass v_h und v_i in derselben Menge des MAX-CUT vorkommen, also der Schnitt nicht durch die Kante e_m geht. In diesem Fall trägt diese Kante für den SPScore je 3 `A`-`T`-Mismatches an k^5 Stellen bei, da die beiden Sequenzen t_h und t_i im **gap-0-1-MSA-SP** nicht gegeneinander verschoben sind.

- $\alpha_{t_h} \neq \alpha_{t_i}$

  Das bedeutet, dass v_h und v_i in verschiedenen Mengen des MAX-CUT vorkommen, also der Schnitt durch die Kante e_m geht. In diesem Fall trägt die Kante für den SPScore je ein A-A-Match, ein T-T-Match und ein A-T-Mismatch an k^5 Stellen bei, da die beiden Sequenzen t_h und t_i im **gap-0-1-MSA-SP** um ein Zeichen gegeneinander verschoben sind:

$$\begin{array}{lccccc} \alpha_{t_h} & \texttt{-TT} & \cdots & \texttt{TTATAT} & \cdots & \texttt{TTT} \\ \alpha_{t_i} & \texttt{TTT} & \cdots & \texttt{TTATTT} & \cdots & \texttt{TT-} \end{array}$$

Die Gaps am Anfang und Ende der Sequenzen tragen insgesamt $z\mathcal{O}(k^4) = \mathcal{O}(k^4)$ zum Gesamtscore bei. Da bei Polynomialzeit-Reduktionen nur große Instanzen von Interesse sind, sei im folgenden sei stets $k > k_0$, wobei k_0 geeignet gewählt sei.

Sei α eine **gap-0-1-MSA-SP**-Instanz der Sequenzen $\bar{t}^G$ und c_α die Anzahl der Kanten des durch das Alignment α gegebenen Schnittes (V_0^α, V_1^α) durch G. In einem **gap-0-1-MSA-SP** α^* der Sequenzen $\bar{t}^G$, das keine Gaps enthält also einfach nur aus dem Übereinanderschreiben der Sequenzen besteht, gibt es insgesamt $3k^5l(k^2-1)$ Mismatches. Also gilt

$$\mathrm{SPScore}(\alpha^*) = \underbrace{(\underbrace{k^{12}lk^2(k^2-1)/2}_{\text{alleZeichen}} - \underbrace{3k^5l(k^2-1)}_{\text{Mismatches}})v_{\mathtt{T}}}_{\text{Matches}} + \underbrace{3k^5l(k^2-1)}_{\text{Mismatches}}u \qquad (2.1)$$

Durch Einfügen der Gaps erhält man daraus dann das Alignment α und durch die entsprechende Scoreveränderung durch das Einfügen der Gaps den $\mathrm{SPScore}(\alpha)$.

$$\begin{aligned} \mathrm{SPScore}(\alpha) &= \mathrm{SPScore}(\alpha^*) - \mathrm{Scoredifferenz} \\ &= \mathrm{SPScore}(\alpha^*) - \underbrace{k^5(2u - v_{\mathtt{A}} - v_{\mathtt{T}})c_\alpha}_{\text{Kanten im Schnitt}} + \underbrace{\mathcal{O}(k^4)}_{\text{Gaps}} \\ &\overset{(2.1)}{=} (k^{12}lk^2(k^2-1)/2 - 3k^5l(k^2-1))v_{\mathtt{T}} + \\ &\quad 3k^5l(k^2-1)u - k^5(2u - v_{\mathtt{A}} - v_{\mathtt{T}})c_\alpha + \mathcal{O}(k^4) \qquad (2.2) \end{aligned}$$

Wegen $u > \max\{0, v_{\mathtt{A}}, v_{\mathtt{T}}\}$ wird für große k dieser Score also genau dann minimal, wenn c_α maximal wird. □

Beweis von Aussage 2: Für den Nachweis der **NP**-Vollständigkeit von **gap-0-MSA-SP** und **MSA-SP** benötigen wir folgendes Lemma, auf dessen Beweis wir hier verzichten:

Lemma 2.7 *Sei $z > \max\{0, v_{\mathtt{T}}\}$ und α eine* **gap-0-MSA-SP**- *oder* **MSA-SP**-*Instanz mit den Sequenzen $\bar{t}^G$. Dann enthalten höchstens $\mathcal{O}(k^6)$ Spalten Gaps.*

Dieses Lemma stellt sicher, dass auch bei diesen Alignment-Arten nur die Codierungen einer Kante aligned werden, da die Codierungen von verschiedenen Kanten weit genug voneinander entfernt sind ($k^7 - 3$ Zeichen). In diesem Fall sind die Mengen des Schnitts V_0^α und V_1^α wie folgt definiert:

$$\begin{aligned} V_0^\alpha &:= \{v_i \in V \;\; \alpha_{t_i} \text{ hat gerade Anzahl von Gaps am Anfang}\} \\ V_1^\alpha &:= V \setminus V_0^\alpha \end{aligned}$$

Dann folgt die Behauptung für den zweiten Fall sogar mit dem gleichen SPScore wie beim **gap-0-1-MSA-SP** (siehe (2.2)). □

Beweis von Aussage 3: Beim Beweis der **NP**-Vollständigkeit des **MSA-SP** treten zwar wegen Lemma 2.7 nicht zu viele Gaps auf, die dann die Strafen für Mismatches ausgleichen könnten, jedoch können sich diese auch innerhalb der Sequenzen befinden. Dazu wird das Alignment α von $\bar{t}^G$ in k^5 Blöcke

$$B_n := \text{Spalten } [k^7 ln] \text{ bis } [k^7 l(n+1) - 1] \text{ von } \alpha$$

eingeteilt. Wegen Lemma 2.7 befinden sich die Codierungen von Kanten im **MSA-SP** im jeweils gleichen Block wie in den nicht angeordneten Sequenzen $\bar{t}^G$. Für diese Blöcke gibt es nun zwei Möglichkeiten, nämlich dass sie Gaps enthalten oder nicht. Beim optimalen Alignment kommen auf jeden Fall Blöcke B_n ohne Gaps vor (zum Beweis siehe [Jus99], S. 9, Fallunterscheidung). In diesen Blöcken sind dann alle Kanten des Graphen codiert und ein optimales Alignment aller solchen Blöcke mit dem minimalen Score liefert den MAX-CUT des Graphen:

Sei U die Menge der Blöcke B_n ohne Gaps und zu jedem Block B_n mit $n \in U$ bezeichne $c_{\alpha,n}$ die Größe des induzierten Schnitts durch den Graphen. Dabei wandelt man die Definition der Mengen V_0^α und V_1^α derart ab, dass man die Gesamtzahl der links von Block B_n eingefügten Gaps betrachtet und ob diese gerade oder ungerade ist. Zum Beweis der Aussage 3 von Satz 2.6 erhält man dann für den SPScore $\text{SPScore}(\alpha)$ einer **MSA-SP**-Instanz folgende Abschätzungen für alle $n \in U$:

$$\begin{aligned} \text{SPScore}(\alpha) &\geq (k^{12} l k^2 (k^2-1)/2 - 3k^5 l(k^2-1)) v_{\mathtt{T}} + 3k^5 l(k^2-1)u \\ &\quad + \text{Scoreverbesserung von } B_n + \mathcal{O}(k^4) \\ &\geq (k^{12} l k^2 (k^2-1)/2 - 3k^5 l(k^2-1)) v_{\mathtt{T}} + 3k^5 l(k^2-1)u \\ &\quad - c_{a,n} k^5 (2u - v_{\mathtt{A}} - v_{\mathtt{T}}) + \mathcal{O}(k^4) \end{aligned} \tag{2.3}$$

Eine **MSA-SP**-Instanz kann keinen schlechteren SPScore haben als eine **gap-0-1-MSA-SP**-Instanz zu den gleichen Sequenzen. Mit c_β als Größe des von einer **gap-0-1-MSA-SP**-Instanz β induzierten MAX-CUT gilt also:

$$\begin{aligned} \text{SPScore}(\alpha) &\leq (k^{12} l k^2 (k^2-1)/2 - 3k^5 l(k^2-1)) v_{\mathtt{T}} + 3k^5 l(k^2-1)u \\ &\quad - c_\beta k^5 (2u - v_{\mathtt{A}} - v_{\mathtt{T}}) + \mathcal{O}(k^4) \end{aligned} \tag{2.4}$$

Daraus und Ungleichung (2.3) folgt $c_\beta \leq c_{\alpha,n}$ und mit der Maximalität von c_β dann $c_\beta = c_{\alpha,n}$ für alle $n \in U$. Zur Entscheidung, ob ein induzierter Schnitt entsprechender Größe existiert, muss man c_β bestimmen. Da dieses denselben Wert wie die $c_{\alpha,n}$ mit $n \in U$ hat und diese wiederum den maximalen Wert unter allen $\{c_{\alpha,n}\}$ haben, wählt man ein maximales $c_{\alpha,n}$ und erhält damit c_β. □

2.1.2 Tree Alignment ist NP-vollständig

Nicht nur das Multiple Alignment mit SPScore ist **NP**-vollständig, sondern auch die andere hier betrachtete Variante, das Tree Alignment (vgl. Abschnitt 1.1.3).

Satz 2.8 *Das* **Tree Alignment-Problem ohne weitere Vorgaben** *ist* **NP**-*vollständig.*

Satz 2.9 *Das* **Tree Alignment-Problem, bei dem die vorgegebene Phylogenie ein binärer Baum ist**, *ist* **NP**-*vollständig.*

2.2 Nichtapproximierbarkeit

2.2.1 Multiple Alignment ist MAXSNP-hard

Um nachzuweisen, dass Multiple Alignment mit SPScore **MAXSNP-hard** ist, wird dieses als Optimierungsproblem definiert:

Definition 2.10 MSA-SP-S

Gegeben:	*Ein endliches Alphabet* Σ, $k > 0$, *Sequenzen* $\{s_1, \ldots, s_k\}$ *über* Σ *und ein Scoring-System (als* $(\lvert\Sigma\rvert + 1) \times (\lvert\Sigma\rvert + 1)$*-Matrix)*
Lösungen:	*Alignment* α *der Sequenzen* $\{s_1, \ldots, s_k\}$
Bewertung:	SPScore(α)
Optimum:	*Minimum*

In dieser Definition wird also das Scoring-System als Teil der Eingabe angesehen, deshalb die Bezeichnung **MSA-SP-S** . Für den Nachweis von **MSA-SP-S**$\in$ **MAXSNP-hard** betrachten wir eine eingeschränkte Version des Problems mit festem Scoring-System. Zu einem vorgegebenen B betrachten wir das in Abbildung 2.3 angegebene Scoring-System:

Definition 2.11 MSA-SP-B

Gegeben:	$k > 0$, *Sequenzen* $\{s_1, \ldots, s_k\}$ *über* $\Sigma_B = \{\mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}\}$
Lösungen:	*Alignment* α *der Sequenzen* $\{s_1, \ldots, s_k\}$
Bewertung:	SPScore(α), *wobei das Scoring-System in Abbildung 2.3 verwendet wird, das von* B *abhängt*
Optimum:	*Minimum*

	C	D	E	F	Δ
C	0	1	0	$4B$	$4B$
D	1	0	0	$4B$	$4B$
E	0	0	0	0	$4B$
F	$4B$	$4B$	0	0	0
Δ	$4B$	$4B$	$4B$	0	0

Abbildung 2.3: Scoring-System von **MSA-SP-B**. Es gehört zum festen Alphabet $\Sigma_B := \{\mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}\}$ und hängt vom Parameter B ab.

Im Folgenden zeigen wir durch Angabe einer L-Reduktion die Existenz eines B, derart dass **MSA-SP-B** in **MAXSNP-hard** liegt. Dabei reduzieren wir das definierte Problem **SIMPLE MAX-CUT-B** mit einer L-Reduktion auf **MSA-SP-B**, wobei B als Konstante so gewählt ist, dass **SIMPLE MAX-CUT-B**$\in$ **MAXSNP-hard** ist.

Definition 2.12 SIMPLE MAX-CUT-B (Optimierungsversion) [2]

Gegeben: *Einfacher, ungerichteter Graph* $G = (V, E)$ *mit* $1 \leq \deg(v) \leq B$ *für alle* $v \in V$

Lösungen: *Partition* $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$

Bewertung: $c(V_1) := |\{\ \{v_1, v_2\} \in E \mid v_1 \in V_1 \wedge v_2 \in V_2\}|$

Optimum: *Maximum*

$c(V_1)$ gibt die Größe eines Schnitts durch den Graphen an, durch den die Knotenmenge und damit der ganze Graph in zwei Teile zerlegt wird; daher die Bezeichnung **SIMPLE MAX-CUT-B**.

Im folgenden wird eine Reduktion angegeben, die zu jedem einfachen Graphen mit durch B beschränktem Knotengrad eine Menge von Sequenzen über dem Alphabet Σ_B festlegt. Aus dem entsprechenden Multiple Alignment der Sequenzen läßt sich dann ein Schnitt durch den Graphen konstruieren. Danach werden wir nachweisen, dass es sich dabei um eine L-Reduktion handelt und erhalten dadurch die Aussage **MSA-SP-B**$\in$ **MAXSNP-hard**.

Nach diesem allgemeinen Teil der Reduktion werden nun zu einer **SIMPLE MAX-CUT-B**-Instanz, also einem einfachen, ungerichteten Graphen $G = (V, E)$, k Sequenzen konstruiert, wobei $k := |V|$. Setze $n := |E|$ und numeriere die Knoten und Kanten von G geeignet: $V = \{v_1, \ldots, v_k\}$, $E = \{e_1, \ldots, e_n\}$. Daraus konstruiere k Sequenzen $s_1, \ldots, s_k$ der Länge $|s_i| = 6n + 5$ $(1 \leq i \leq k)$ wie folgt:

1. Setze $s_i[1] := \mathtt{F}$ und $s_i[6n + 5] := \mathtt{F}$.

[2] Auch hier ändert die zusätzlich getroffene Annahme $1 \leq \deg(v)$ nicht die Schwierigkeit des Verfahrens, da die Reduktionen zum Nachweis der Schwierigkeit von **SIMPLE MAX-CUT-B** ohne isolierte Knoten auskommen.

2. Für jede Kante $e_m = \{v_i, v_j\} \in E$ (ohne Einschränkung sei dabei $i < j$) setze

$$\begin{array}{lll} s_i[6m-2] := \mathtt{C}, & s_i[6m-1] := \mathtt{D}, & s_i[6m] := \mathtt{C} \quad \text{und} \\ s_j[6m-2] := \mathtt{D}, & s_j[6m-1] := \mathtt{C}, & s_j[6m] := \mathtt{D} \end{array}$$

3. In allen anderen Fällen setze $s_i[j] := \mathtt{E}$.

Falls also beispielsweise die Kante $e_m = v_i, v_j$ in G vorkommt, so tritt folgende Codierung in den Sequenzen s_i und s_j auf:

$$\begin{array}{lccccc} s_i & \mathtt{FE} & \cdots & \mathtt{EECDCEE} & \cdots & \mathtt{EF} \\ s_j & \mathtt{FE} & \cdots & \mathtt{EEDCDEE} & \cdots & \mathtt{EF} \\ & & & \uparrow & & \\ & & & 6m-1 & & \end{array}$$

Werden diese beiden Sequenzen um ein Zeichen gegeneinander verschoben, so entfallen die drei $\mathtt{C}-\mathtt{D}$-Mismatches und werden durch Matches bzw. Mismatches mit dem Zeichen E ersetzt, die alle den Score 0 haben. Also verbessert sich der Score, für eine Kante e_m wie oben:

$$\begin{array}{lccccc} \alpha_{s_i} & \mathtt{-FE} & \cdots & \mathtt{EECDCE} & \cdots & \mathtt{EEF} \\ \alpha_{s_j} & \mathtt{FEE} & \cdots & \mathtt{EDCDEE} & \cdots & \mathtt{EF-} \end{array}$$

Auf dieser Beobachtung baut nun die Bestimmung des Schnitts (V_1, V_2) im Graphen auf. Das folgende Lemma 2.13 gewährleistet, dass in einem optimalen Alignment keine Gaps vorkommen, die mit $4B$ bestraft werden. Im wesentlichen können wir also aus einem optimalen Alignment ähnlich im Falle des **gap-0-1-MSA-SP** im Beweis von Satz 2.6 die beiden Mengen V_1 und V_2 des Schnitts rekonstruieren, indem wir gegeneinander verschobene Sequenzen als zu unterschiedlichen Punktmengen gehörend definieren. Leider gibt es aber noch Gaps, die nicht bestraft werden und gerade diese machen eine etwas andere Definition notwendig. Beispielsweise kann ein optimales Alignment am Anfang der Sequenzen folgendes Aussehen aufweisen:

$$\begin{array}{ll} \alpha_{s_1} & \mathtt{F--EE}\cdots\cdots \\ \alpha_{s_2} & \mathtt{-F-EE}\cdots\cdots \\ \alpha_{s_3} & \mathtt{--FEE}\cdots\cdots \\ \alpha_{s_4} & \mathtt{---FE}\cdots\cdots \\ \alpha_{s_5} & \mathtt{F--EE}\cdots\cdots \end{array}$$

Diese Gaps ändern nichts an der Bewertung des Alignments und auch nichts daran, wie die Sequenzen um ein Zeichen gegeneinander verschoben sind. Deshalb wird einfach die erste Spalte des Alignments verwendet, in der das Zeichen E auftritt, um den Schnitt durch den Graphen zu definieren. Hat diese Spalte den Index j, so setze:

$$\begin{array}{lcl} V_1 & := & \{v_i \mid \alpha_{s_i}[j] = \mathtt{E}\} \\ V_2 & := & V \setminus V_1 \end{array}$$

Das folgende Lemma liefert nun die Tatsache, dass es sich in einem optimalen Alignment bei den Sequenzteilen, die kein F enthalten, wirklich um eine **gap-0-1-MSA-SP**-Instanz handelt, nur dass in diesem Fall die Gaps als F dargestellt werden.

Lemma 2.13 *Kommen in einem Alignment α der oben konstruierten Sequenzen $\{s_1, \ldots, s_k\}$ Gaps vor, die mit $4B$ bestraft werden, so gilt:*

$$\text{SPScore}(\alpha) > 3n$$

Jedes Alignment mit einem niedrigen SPScore enthält also keine Gaps, die mit $4B$ zum Score beitragen, insbesondere keine internen Gaps. Zum Beweis gehen wir vom trivialen Alignment α^* aus, das gar keine Gaps enthält, also einfach dem Übereinanderschreiben der gleich langen Sequenzen $\{s_1, \ldots, s_k\}$ entspricht. Jede Kantencodierung erzeugt in diesem Alignment 3 Mismatches, also gilt:

$$\text{SPScore}(\alpha^*) = 3n$$

Liegt nun ein Alignment α vor, das Gaps enthält, so kann man dieses aus α^* durch sukzessives Einfügen von Gaps gewinnen. Wird nun ein in α vorhandender Gap der Bewertung $4B$ in α^* eingefügt, so erhöht dies den Score des Gesamtalignments um $4B$. Andererseits fallen in der Sequenz, in der der Gap eingefügt wird, bis zu $3B$ Mismatches des Gesamtalignments weg, was einer Scoreverbesserung von bis zu $3B$ entspricht.

Dies folgt aus den Tatsachen, dass jeder Knoten im Graphen G höchstens den Grad B hat und dass Mismatches mit dem Zeichen E nicht bestraft werden. Durch die gewählte Codierung von Kanten kann die zu einem Knoten gehörige Sequenz also höchstens $3B$ Mismatches zum Gesamtscore beitragen.

Das Einfügen eines Gaps mit Bestrafung $4B$ aus α in α^* wirkt sich also wie folgt auf den Score aus:

$$\text{SPScore}(\alpha) \geq \text{SPScore}(\alpha^*) + 4B - 3B = \text{SPScore}(\alpha^*) + B = 3n + B > 3n$$

□

Mit dieser Aussage ist es nun möglich, die Reduktion anzugeben: R und S sind ein Paar von Abbildungen, wobei R zu jedem Graphen die Sequenzen $\{s_1, \ldots, s_k\}$ und S zu jedem Alignment solcher Sequenzen den Schnitt (V_1, V_2) bestimmt. Da dies mit den in Definition 1.21 zugelassenen Ressourcen möglich ist, müssen nur noch die Auswirkungen der Funktionen R und S auf die Bewertungsfunktionen der beiden Probleme betrachtet werden, um nachzuweisen, dass (R, S) eine L-Reduktion von **SIMPLE MAX-CUT-B** auf **MSA-SP-S** ist.

Sei mit $c(V_1)$ die Bewertungsfunktion des **SIMPLE MAX-CUT-B** bezeichnet (vergleiche Definition 2.12). Diese gibt also die Anzahl der Kanten im Schnitt (V_1, V_2) an. Mit $c^*(\alpha) := \text{SPScore}(\alpha)$ sei die Kostenfunktion von **MSA-SP-S** bezeichnet. Dann gilt für ein optimales Alignment α der zu den Knoten

von G gehörenden Sequenzen und den daraus berechneten Schnitt der Größe c_α:

$$c^*(\alpha) = \text{SPScore}(\alpha) = 3(n - c_\alpha)$$

Weiter gilt in jedem beliebigen, ungerichteten, einfachen Graphen $G = (V, E)$ mit $1 \leq \deg(v) \leq B$ für alle $v \in V$, für die Größe $\text{OPT}(G)$ eines maximalen Schnittes:

$$\text{OPT}(G) \geq \frac{|E|}{B} \tag{2.5}$$

Dies folgt aus der Tatsache, dass bei einem maximalen Schnitt für jeden Knoten v mit $deg(v) \geq 1$ mindestens eine adjazente Kante im Schnitt liegen muss. Ansonsten könnte man einen Knoten, für den das nicht gilt, in die andere der Mengen V_1 und V_2 verschieben, was eine Vergrößerung des Schnitts zur Folge hätte. Damit folgt $\text{OPT}(G) \geq \frac{|V|}{2}$ und mit der Abschätzung $|V| \geq \frac{2|E|}{B}$ die Ungleichung (2.5).

Satz 2.14 $\mathbf{MSA - SP - S \in MAXSNP{-}hard}$

Dazu reicht es aus, folgende Aussage zu zeigen:
(R, S) ist eine L-Reduktion mit den Parametern $\beta := 3B$ und $\gamma := \frac{1}{3}$.
Aus $\mathbf{SIMPLEMAX - CUT - B \in MAXSNP{-}hard}$ und obigen Überlegungen folgt dann die Aussage.
Nach Definition 1.21 müssen diese beiden Ungleichungen erfüllt sein:

1. Für jeden einfachen, ungerichteten Graphen G gilt:
$$\text{OPT}(R(G)) \leq \beta \text{OPT}(G)$$
2. Für jeden einfachen, ungerichteten Graphen G und jedes Alignment α der Sequenzen $R(G)$ gilt:
$$|\text{OPT}(G) - c(S(\alpha))| \leq \gamma |\text{OPT}(R(G)) - c^*(\alpha)|$$

Beweis von 1. und 2.:

1.
$$\text{OPT}(R(G)) = 3(n - \text{OPT}(G)) \leq 3n = 3B\frac{n}{B} \overset{(2.5)}{\leq} 3B\,\text{OPT}(G) = \beta \text{OPT}(G)$$

2. Für den Nachweis der Gültigkeit der zweiten Ungleichung sind zwei Fälle zu betrachten:

   - 1. Fall: $c^*(\alpha) = \text{SPScore}(\alpha) \leq 3n$
     Nach Lemma 2.13 wird hier ein Schnitt der Größe c_α induziert und es gilt: $c^*(\alpha) = 3(n - c_\alpha)$.

$$|\text{OPT}(G) - c(S(\alpha))| = |\text{OPT}(G) - c_\alpha| = \frac{1}{3}|\,3c_\alpha - 3\text{OPT}(G)|$$

$$= \frac{1}{3}|3(n - \text{OPT}(G)) - 3(n - c_\alpha)| = \frac{1}{3}|\text{OPT}(R(G)) - c^*(\alpha)|$$

$$= \gamma |\text{OPT}(R(G)) - c^*(\alpha)|$$

- 2. Fall: $3n + c := c^*(\alpha) > 3n$
  Die Größe des durch das Alignment induzierten Schnitts sei mit x bezeichnet und offensichtlich gilt: $0 \leq x \leq \mathrm{OPT}(G)$. Dann ist auch die zweite Ungleichung erfüllt:

$$|\mathrm{OPT}(G) - c(S(\alpha))| = \mathrm{OPT}(G) - x \leq \mathrm{OPT}(G) \leq \frac{1}{3}|3\mathrm{OPT}(G) + c|$$

$$= \frac{1}{3}|3n + c - 3n + 3\mathrm{OPT}(G)| = \frac{1}{3}|(3n + c) - 3(n - \mathrm{OPT}(G))|$$

$$= \frac{1}{3}|c^*(\alpha) - \mathrm{OPT}(R(x)))| = \gamma|\mathrm{OPT}(R(x))) - c^*(\alpha)|$$

□

Korollar 2.15 MSA-SP-S *ist* **MAXSNP-hard**

Hier wird bei der Reduktion von **MSA-SP-B** auf **MSA-SP-S** zusätzlich das fest vorgegebene Alphabet und Scoringsystem von **MSA-SP-B** in die Eingabe von **MSA-SP-S** aufgenommen. □

2.2.2 Tree Alignment ist MAXSNP-hard

Satz 2.16 *Tree Alignment ist* **MAXSNP-hard**

Zum Beweis dieser Aussage wird das Problem **VERTEX-COVER-B** als Ausgangspunkt einer L-Reduktion auf das unten definierte **Eingeschränkte Tree Alignment-Problem** verwendet. Für **VERTEX-COVER-B** gibt es ein B, so dass es **MAXSNP-hard** ist und es ist folgendermassen definiert:

Definition 2.17 VERTEX-COVER-B[3]

Gegeben:	*Ungerichteter Graph* $G = (V, E)$ *mit* $deg(v) \leq B \ \forall v \in V$, $K > 0$
Lösungen:	$V' \subseteq V$ *derart, dass für alle* $\{u, v\} \in E$ *gilt:* $u \in V'$ *oder* $v \in V'$
Bewertung:	$c(V') = \lvert V' \rvert$
Optimum:	*Minimum*

Definition 2.18 Eingeschränktes Tree Alignment-Problem

Gegeben:	*Mengen* X *und* Y *über einem endlichen Alphabet* Σ
Lösungen:	$Y' \subseteq Y$ *und ein phylogenetischer Baum* $T_{X,Y'}$, *bei dem die Blätter genau mit den Sequenzen aus* X *und die anderen Knoten mit den Sequenzen aus* Y' *markiert sind.*
Bewertung:	*Score von* $T_{X,Y'}$
Optimum:	*Minimum*

[3]Das Problem **VERTEX-COVER-B** wird in der Literatur auch oft als **NODE-COVER-B** bezeichnet

Da dieses Problem mit einer L-Reduktion auf Tree Alignment reduziert werden kann, folgt dann die Aussage.

Satz 2.19 *Die Konstruktion eines Tree Alignment bei dem der vorgegebene phylogenetische Baum ein Stern*[4] *ist, ist* **MAXSNP−hard.**

Zum Beweis wird hier **MAX-CUT-B** auf diesen Spezialfall des Tree Alignment reduziert. **MAX-CUT-B** ist die Version des in Definition 2.1 vorgestellten **SIMPLE MAX-CUT-B** für nicht notwendigerweise einfache Graphen mit beschränktem Knotengrad. Ein Beispiel für ein **Tree Alignment mit Stern als Phylogenie** ist in Abbildung 2.4 zu finden.

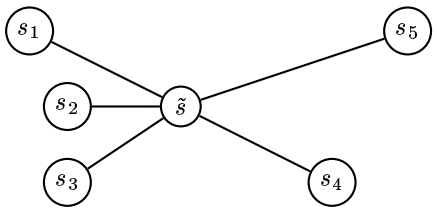


Abbildung 2.4: Tree Alignment mit Stern-Phylogenie für 5 Sequenzen $s_1, \ldots, s_5$; $\tilde{s}$ ist der String, der den Gesamtscore des Tree Alignment minimiert.

2.3 Approximierbarkeit

2.3.1 1/2-Approximations-Algorithmus für Multiple Alignment

Im Folgenden werden wir einen Algorithmus vorstellen, der in polynomieller Zeit für k Sequenzen eine Näherungslösung eines optimalen Multiple Alignment bestimmt, die nur um den Faktor $2 - \frac{2}{k}$ schlechter ist als der optimale Score. Daraus ergibt sich für den relativen Fehler dann eine obere Schranke von $1 - \frac{1}{2-\frac{2}{k}}$, die durch $\frac{1}{2}$ nach oben abgeschätzt werden kann und damit einen $1 - \frac{1}{2-\frac{2}{k}}$- bzw. $\frac{1}{2}$-Approximations-Algorithmus liefert.

Dabei wird eine Sequenz ausgewählt, die zu allen anderen Sequenzen die kleinste paarweise Distanz hat, und an dieser durch sukzessives Hinzufügen der anderen Sequenzen ein Multiple Alignment aufgebaut. Die Wahl einer Sequenz und die Berechnung der Distanzen zu den anderen Sequenzen läßt sich als sternförmiger Graph darstellen, bei dem die Ausgewählte Sequenz dem Knoten in der Mitte zugeordnet ist und die anderen Sequenzen den anderen Knoten. Ein Beispiel ist in 2.6 angegeben. Daher kommt auch die Bezeichnung „Center Star Alignment".

[4]Dieses Tree Alignment gehört zu der in Definition 1.10 beschriebenen Klasse und ist in Abschnitt 2.3.2 genauer definiert, während in Definition 1.11 der Begriff „Stern" festgelegt ist.

Definition 2.20 *Sei $S = \{s_1, \ldots, s_k\}$ eine Menge von Sequenzen. Wie in Definition 1.3 wird mit* $\mathrm{sc}(s_i, s_j)$ *die paarweise Distanz von zwei Sequenzen s_i und s_j bezeichnet, also der Score eines optimalen paarweisen Alignments von s_i und s_j.*

Weiter erfülle $\mathrm{sc}(\cdot,\cdot)$ *die Dreiecksungleichung (1.6), sei positiv definit (also* $\mathrm{sc}(\sigma_1, \sigma_2) \geq 0 \ \forall \sigma_1, \sigma_2 \in \tilde{\Sigma}$*) sowie* $\mathrm{sc}(\Delta, \Delta) = 0$*, wobei Δ wieder für ein Gap steht und das Scoring-System sei symmetrisch:*

$\forall \sigma_1, \sigma_2 \in \tilde{\Sigma}\ \mathrm{sc}(\sigma_1, \sigma_2) = \mathrm{sc}(\sigma_2, \sigma_1)$.

Sei $\alpha = (\alpha_{s_1}, \ldots, \alpha_{s_k})$ ein optimales SPScore*-Multiple Alignment der Sequenzen $s_1, \ldots, s_k$, wobei für alle i die Sequenz α_{s_i} wie in Definition 1.7 aus s_i entsteht.*

Dann bezeichne $\mathrm{d}(\alpha_{s_i}, \alpha_{s_j})$ *die induzierte Distanz der Projektion $(\alpha_{s_i}, \alpha_{s_j})$ des Alignments α auf die Sequenzen α_{s_i} und α_{s_j}:*

$$\mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) := \sum_{l=1}^{|\alpha_{s_1}|} \mathrm{sc}(\alpha_{s_i}[l], \alpha_{s_j}[l]) \tag{2.6}$$

Weiter sei mit $\mathrm{sc}(\alpha) := \sum_{i<j} \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) = \mathrm{SPScore}(\alpha)$ *der* SPScore *von α bezeichnet.*

Da α zusätzliche Gaps und damit auch Mismatches enthalten kann, die im optimalen paarweisen Alignment zweier Sequenzen s_i und s_j nicht vorkommen, gilt:

$$\mathrm{sc}(s_i, s_j) \leq \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}). \tag{2.7}$$

Definition 2.21 Induziertes Multiple Alignment

Sei $S = \{s_1, \ldots, s_k\}$ eine Menge von Sequenzen und $T = (V, E)$ ein Baum, bei dem jeder Knoten v mit einer Sequenz $s_v \in S$ versehen ist. Ein Multiple Alignment α von S heißt dann induziert von T, falls $\forall v, v' \in V$ gilt:

$$(v, v') \in E \ \Rightarrow \ \mathrm{d}(\alpha_{s_v}, \alpha_{s_{v'}}) = \mathrm{sc}(s_v, s_{v'})$$

Mit der in Definition 2.20 aufgeführten Bedingung $\mathrm{sc}(\Delta, \Delta) = 0$, die für Multiple Alignment mit SPScore notwendig ist, kann man aus beliebigen Tree Alignments ein induziertes Alignment gewinnen.

In Abb. 2.5 ist ein Beispiel für ein induziertes Multiple Alignment aufgeführt. Setzt man dort etwa $\mathrm{sc}(\sigma, \sigma) = 0$, $\mathrm{sc}(\sigma_1, \sigma_2) = 2$, $\mathrm{sc}(\sigma, \Delta) = \mathrm{sc}(\Delta, \sigma) = 3$ und $\mathrm{sc}(\Delta, \Delta) = 0 \quad \forall \sigma, \sigma_1, \sigma_2 \in \Sigma$ mit $\sigma_1 \neq \sigma_2$, so ergibt sich für alle i:

$$\mathrm{d}(\alpha_{s_1}, \alpha_{s_i}) = \mathrm{sc}(s_1, s_i),$$

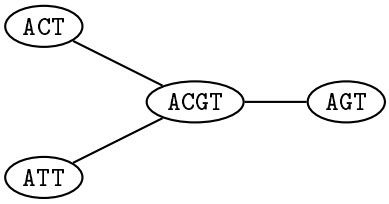
wie gefordert.

Definition 2.22 Center Star

Sei $S = \{s_1, \ldots, s_k\}$ Menge von Sequenzen, $M_i := \sum_{j=1}^{k} \mathrm{sc}(s_j, s_i)$.

$c \in 1, \ldots, k$ bezeichne dann den Wert, der M_i minimiert:

$$M_c = \min\{M_i \mid 1 \leq i \leq k\}$$



$$\begin{array}{lcccc} \alpha_{s_1} = & \mathtt{A} & \mathtt{C} & \mathtt{G} & \mathtt{T} \\ \alpha_{s_2} = & \mathtt{A} & \mathtt{C} & \mathtt{-} & \mathtt{T} \\ \alpha_{s_3} = & \mathtt{A} & \mathtt{-} & \mathtt{T} & \mathtt{T} \\ \alpha_{s_4} = & \mathtt{A} & \mathtt{-} & \mathtt{G} & \mathtt{T} \end{array}$$

Abbildung 2.5: Beispiel für Tree Alignment und ein induziertes Multiple Alignment

Der Center Star ist dann ein Tree Alignment mit Stern T als Baum, bei dem die Wurzel mit s_c markiert ist und die k Blätter mit den Sequenzen aus S.

Das induzierte Multiple Alignment wird dann mit $\alpha^c = (\alpha^c_{s_1}, \ldots, \alpha^c_{s_k})$ bezeichnet.

Der Gesamtscore $\mathrm{sc}(\alpha^c)$ *von* α^c *ist wie folgt definiert:*

$$\mathrm{sc}(\alpha^c) := \mathrm{SPScore}(\alpha^c) = \sum_{i<j} \mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_j})$$

In Abbildung 2.6 ist ein Beispiel mit 5 Sequenzen abgebildet.

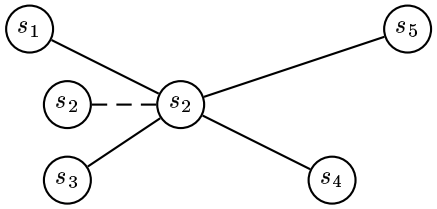


Abbildung 2.6: Center Star Alignment mit 5 Sequenzen und $s_c = s_2$. Die gestrichelte Kante hat das Gewicht (bzw. die Länge) 0

Wie im folgenden Satz 2.24 noch gezeigt werden wird, ist das Center-Star-Alignment eine Näherungslösung für ein optimales Alignment mit relativem Fehler $1 - \frac{1}{2-\frac{2}{k}}$. Mit dem Algorithmus 2.1 läßt es sich effizient berechnen, weshalb dann dieser Algorithmus ein $1 - \frac{1}{2-\frac{2}{k}}$-Approximations-Algorithmus für Multiple Alignment ist.

Dabei wird nach der Bestimmung von c die Sequenz s_c zum Aufbau des Multiple Alignment α^c verwendet, indem zu jeder neuen Sequenz s_i ein Alignment mit dieser Sequenz s_c berechnet wird. Dazu dient die Prozedur *pwalign*,

Algorithmus 2.1 Center Star Multiple Alignment

Eingabe: Sequenzen $S = \{s_1, \ldots, s_k\}$
Ausgabe: Center-Star induziertes Multiple Alignment $\alpha^c = (\alpha^c_{s_1}, \ldots, \alpha^c_{s_k})$

```
                                                    {bestimme c}
1:  c := 1                                          {Startwert für min-Berechnung}
2:  M := Σ_j sc(s_j, s_1)                           {sc(s_j, s_1) wie in Kap. 1.1.1 berechnet}
3:  for i := 2 to k do
4:     M' := Σ_j sc(s_j, s_i)
5:     if M' < M then                               {neues min gefunden?}
6:        M := M'; c := i
                {bestimme α^c, induziert aus Center Star mit oben berechnetem c}
7:  α^c_{s_c} := s_c                                {Initialisierung; an α^c_{s_c} wird α^c aufgebaut}
8:  for i := 1 to k do
9:     if i ≠ c then                                {α^c_{s_c} nur einmal in α^c}
10:       (α^c_{s_c}, α^c_{s_i}) := pwalign(α^c_{s_c}, α^c_{s_i})   {pwalign(s,t) = (α_s, α_t) paarw.}
                                                    {Alignment wie in 1.1.1 }
11: return (α^c_{s_1}, ..., α^c_{s_k})
```

die ein optimales paarweises Alignment berechnet. In die Sequenz $\alpha^c_{s_c}$ kommen im Schritt 10 neue Gaps und damit in alle bisher schon angeordneten Sequenzen $\alpha^c_{s_1}, \ldots, \alpha^c_{s_{i-1}}$. Einmal in $\alpha^c_{s_c}$ eingefügte Gaps bleiben in allen weiteren Schritten erhalten, kurz: Einmal Gap - immer Gap. Ein Beispiel ist in Abb. 2.7 zu finden.

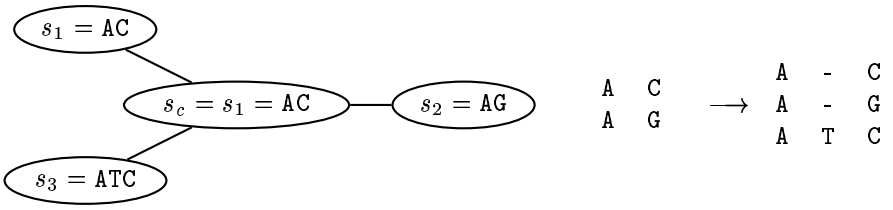


Abbildung 2.7: Beispiel, wie aus einem Center Star ein induziertes Multiple Alignment konstruiert wird. Dargestellt ist auch das Hinzunehmen der Sequenz s_3 zum schon konstruierten Alignment der Sequenzen s_1 und s_2

Laufzeitbetrachtung des Algorithmus 2.1: Nach 1.1.1 benötigt die Berechnung von $\mathrm{sc}(s_i, s_j)$ und $\mathrm{pwalign}(s_i, s_j)$ $\mathcal{O}(n^2)$ Schritte, wobei $n = \max_i |s_i|$ ist. Dies wird in Zeile 2 einmal und in Zeile 4 $(k-2)$-mal ausgeführt, sowie in Zeile 10 $(k-1)$-mal. Insgesamt benötigt man also $\mathcal{O}(2(k-1)n^2) = \mathcal{O}(n^2 k)$ Schritte, was auch gegenüber den $\mathcal{O}(k^2 2^k n^k)$ Schritten des dynamischen Programmierens in Definition 1.8 eine große Beschleunigung darstellt.

Abschließend erfolgt noch die Betrachtung des relativen Approximationsfehlers:

Für s_c gilt nach Konstruktion:

$$\begin{array}{rlll} & \mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_c}) & = & \mathrm{sc}(s_i, s_c) \\ \Rightarrow & \mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_j}) & \overset{\text{Dreiecksungl.}}{\leq} & \mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_c}) + \mathrm{d}(\alpha^c_{s_c}, \alpha^c_{s_j}) \\ & & = & \mathrm{sc}(s_i, s_c) + \mathrm{sc}(s_c, s_j) \end{array} \tag{2.8}$$

Lemma 2.23 *Es gilt:* $\mathrm{sc}(\alpha^c)/\mathrm{sc}(\alpha) \leq 2 - \frac{2}{k} \leq 2$

Zum Beweis betrachten wir $v(\alpha^c) := \sum_{(i,j)} \mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_j}) = 2\,\mathrm{sc}(\alpha^c)$ und $v(\alpha) := \sum_{(i,j)} \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) = 2\,\mathrm{sc}(\alpha)$. Dann gilt:

$$\begin{array}{rll} v(\alpha^c) & = & \displaystyle\sum_{(i,j)} \mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_j}) \\ & \overset{(2.8)}{\leq} & \displaystyle\sum_{(i,j)} [\mathrm{sc}(s_i, s_c) + \mathrm{sc}(s_c, s_j)] \\ & \overset{\text{Symmetrie von sc}(\cdot,\cdot)}{=} & \displaystyle\sum_{(i,j)} \mathrm{sc}(s_i, s_c) + \sum_{(i,j)} \mathrm{sc}(s_j, s_c) \\ & \overset{\mathrm{d}(s,s)=0}{=} & \displaystyle\sum_i \sum_{j \neq i} \mathrm{sc}(s_i, s_c) + \sum_j \sum_{i \neq j} \mathrm{sc}(s_j, s_c) \\ & = & \displaystyle(k-1) \sum_i \mathrm{sc}(s_i, s_c) + (k-1) \sum_j \mathrm{sc}(s_j, s_c) \\ & = & \displaystyle 2(k-1) \sum_i \mathrm{sc}(s_i, s_c) = 2(k-1) M_c \end{array}$$

und:

$$\begin{array}{rll} v(\alpha) & = & \displaystyle\sum_{(i,j)} \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) \\ & \overset{(2.7)}{\geq} & \displaystyle\sum_{(i,j)} \mathrm{sc}(s_i, s_j) \\ & \overset{\text{Wahl von } c}{\geq} & \displaystyle k \sum_j \mathrm{sc}(s_c, s_j) = k M_c \end{array}$$

Mit diesen Ungleichungen folgt:

$$\frac{\mathrm{sc}(\alpha^c)}{\mathrm{sc}(\alpha)} = \frac{v(\alpha^c)}{v(\alpha)} \leq \frac{2(k-1)M_c}{kM_c} = 2 - \frac{2}{k}$$

□

Dieses Lemma gibt einen direkten Einblick in die Größenverhältnisse der Scores der beiden Lösungen. Um jedoch die Ergebnisse dieses Abschnittes in einem Satz zusammenzufassen, verwenden wir folgende Umformung der Aussage:

$$\mathrm{sc}(\alpha) \geq \frac{\mathrm{sc}(\alpha^c)}{2 - \frac{2}{k}} \tag{2.9}$$

Satz 2.24 *Center-Star-Alignment ist ein* $1 - \frac{1}{2-\frac{2}{k}}$*-Approximations-Algorithmus für* **Multiple Alignment**

Dazu betrachten wir den relativen Fehler $R(\{s_1, \ldots, s_k\}, \alpha^c)$ aus Definition 1.16. Dabei sei α ein optimales Alignment der vorgegebenen Sequenzen $\{s_1, \ldots, s_k\}$ und α^c ein dazu konstruiertes Center-Star-Alignment.

$$\begin{aligned} R(\{s_1, \ldots, s_k\}, \alpha^c) &= \frac{|\mathrm{sc}(\alpha^c) - \mathrm{sc}(\alpha)|}{\max\{\mathrm{sc}(\alpha), \mathrm{sc}(\alpha^c)\}} = \frac{\mathrm{sc}(\alpha^c) - \mathrm{sc}(\alpha)}{\mathrm{sc}(\alpha^c)} \\ &\overset{(2.9)}{\leq} \frac{\mathrm{sc}(\alpha^c) - \frac{\mathrm{sc}(\alpha^c)}{2-\frac{2}{k}}}{\mathrm{sc}(\alpha^c)} = 1 - \frac{1}{2 - \frac{2}{k}} \end{aligned}$$

□

Korollar 2.25 *Center-Star-Alignment ist ein* $\frac{1}{2}$*-Approximations-Algorithmus für Multiple Alignment*

Dies folgt aus Satz 2.24 und folgender Abschätzung:

$$\frac{2}{k} \geq 0 \quad \Longrightarrow \quad \frac{1}{2 - \frac{2}{k}} \geq \frac{1}{2} \quad \Longrightarrow \quad 1 - \frac{1}{2 - \frac{2}{k}} \leq \frac{1}{2}$$

□

2.3.2 1/2-Approximations-Algorithmus für Tree Alignment mit Stern-Phylogenie

Die im vorherigen Abschnitt vorgestellte Heuristik eignet sich auch als Approximation des in Satz 2.19 vorgestellten **Tree Alignments mit Stern-Phylogenie**. Mit den im vorherigen Abschnitt durchgeführten Laufzeitbetrachtungen ist mit Algorithmus 2.1 ein polynomiell zeitbeschränkter Approximationsalgorithmus für **Tree Alignment mit Stern-Phylogenie** angegeben. Dabei kann auf die Bestimmung eines optimalen Multiple Alignment verzichtet werden, da nur der in die Mitte gestellte String s_c bzw. dessen Index c von Interesse ist. Den erreichbaren Approximationsfaktor $\varepsilon = \frac{1}{2}$ liefern das folgende Lemma 2.26 und der Satz 2.27.

Seien wieder alle Voraussetzungen von Definition 2.20 erfüllt. Sei $T = (V, E)$ ein Tree Alignment mit Stern-Phylogenie, also ein Baum mit Wurzel v_0 und Blättern v_i, $i = 1, \ldots, k$ und die Kanten des Baums $E = \{(v_r, v_i) \,|\, i = 1, \ldots, k\}$.

Die Blätter seien mit $s_1, \ldots, s_k$ markiert und für jeden beliebigen String $s \in \tilde{\Sigma}^*$, $\tilde{\Sigma} = \Sigma \cup \{\Delta\}$, als Markierung der Wurzel, sei M_s wie folgt definiert:

$$M_s := \sum_j \mathrm{sc}(s_j, s)$$

Dann bezeichnet $\tilde{s} \in \tilde{\Sigma}^*$ diejenige Markierung der Wurzel, für die gilt:

$$M_{\tilde{s}} = \min\{M_s \mid s \in \tilde{\Sigma}^*\}$$

$\mathrm{sc}(\alpha(T)) := M_{\tilde{s}}$ ist dann der Score des Tree Alignments $\alpha(T)$ und offensichtlich gilt $M_{\tilde{s}} \leq M_c$, wobei M_c wie in Definition 2.22 definiert ist.

Lemma 2.26 *Es gilt:* $M_c/M_{\tilde{s}} \leq 2$

Sei $i \in \{1, \ldots, k\}$ so gewählt, dass $\mathrm{sc}(s_i, \tilde{s}) \leq \mathrm{sc}(s_j, \tilde{s})$ für alle $j \in \{1, \ldots, k\}$. Dann gilt:

$$\begin{aligned} M_c &\overset{\text{Def. 2.22}}{\leq} M_i \overset{\text{Def. 2.22}}{=} \sum_j \mathrm{sc}(s_j, s_i) \\ &\overset{\text{Dreiecksungl.}}{\leq} \sum_j (\mathrm{sc}(s_j, \tilde{s}) + \mathrm{sc}(\tilde{s}, s_i)) = \sum_j (\mathrm{sc}(s_j, \tilde{s}) + \mathrm{sc}(s_i, \tilde{s})) \\ &\overset{\text{Wahl von } i}{\leq} 2 \sum_j \mathrm{sc}(s_j, \tilde{s}) = 2M_{\tilde{s}} \Rightarrow \quad M_c/M_{\tilde{s}} \leq 2 \end{aligned}$$

□

Satz 2.27 Tree Alignment mit Stern-Phylogenie *ist* $\frac{1}{2}$*-approximierbar*

Zum Beweis betrachten wir den relativen Fehler:

$$\begin{aligned} R(\{s_1, \ldots, s_k\}, \alpha^c) &= \frac{|M_c - M_{\tilde{s}}|}{\max\{M_{\tilde{s}}, M_c\}|} = \frac{M_c - M_{\tilde{s}}}{M_c} \\ &\leq \frac{M_c - \frac{1}{2}M_c}{M_c} = \frac{\frac{1}{2}M_c}{M_c} = \frac{1}{2} \end{aligned}$$

□

Der in 2.27 bewiesene Approximationsfaktor $\frac{1}{2}$ ist der beste, der sich mit diesem Verfahren erreichen läßt. Der folgende Satz 2.29 sichert zu, dass der relative Fehler des Center Star-Algorithmus beliebig nahe an diesen Faktor herankommt. Das Scoring-System muss dazu wieder die Eigenschaften in Definition 2.20 und außerdem folgende Bedingungen erfüllen:

1. $\mathrm{sc}(\sigma, \sigma) = 0 \quad \forall \sigma \in \Sigma$
2. $\forall \sigma_1, \sigma_2 \in \Sigma \quad 2\,\mathrm{sc}(\sigma_1, \sigma_2) \leq \mathrm{sc}(\sigma_1, \Delta)$

	Δ	A	B
Δ	0	2	2
A	2	0	1
B	2	1	0

Abbildung 2.8: Scoring-System für den Beweis von Satz 2.29

Diese Bedingungen werden von fast jedem in der Praxis verwendeten Scoring-System erfüllt und stellen keine große Einschränkung dar. Zum Beweis des folgenden Satzes wird o. B. d. A. $\Sigma = \{\mathtt{A}, \mathtt{B}\}$ und ein Scoringsystem wie in Abbildung 2.8 angenommen.

Lemma 2.28 *Sei $\varepsilon > 0$. Dann existiert eine Sequenzmenge S mit:*

$$M_c / M_{\tilde{s}} > 2 - \varepsilon$$

Setze $k := n := \lfloor \frac{2}{\varepsilon} \rfloor + 1$.
Konstruiere nun die Menge $S = \{s_1, \ldots, s_k\}$ mit $|s_i| = n$ für alle i wie folgt:

$$s_i[j] := \begin{cases} \mathtt{A}, & \text{falls } i = j \\ \mathtt{B}, & \text{sonst} \end{cases}$$

Also: $s_i =$ A ⋯ A B A ⋯ A
$\uparrow$ $\uparrow$ $\uparrow$
1 i n

Dann gilt für die paarweisen Alignments und einen String $\tilde{s}$ aus einem optimalen Tree Alignment mit Stern-Phylogenie:

1. Kein Alignment $(\alpha_{s_i}, \alpha_{s_j})$ enthält Gaps und für jedes i ist

$$M_i = 2(n-1)$$

2. $\tilde{s} = \mathtt{A}^n$. $(\alpha_{s_i}, \alpha_{\tilde{s}})$ enthält für kein i Gaps und

$$M_{\tilde{s}} = n$$

Beweis von 1:
Für $i \neq j$ sei mit g die Anzahl der Zuordnungen (Δ, σ) bzw. (σ, Δ) im Alignment $\alpha := (\alpha_{s_i}, \alpha_{s_j})$ bezeichnet. Nun betrachten wir die Auswirkungen auf die Scores, falls Gaps auftreten und erhalten damit eine Aussage über die Anzahl g von Gaps in einem optimalen Alignment. Es sind vier Fälle zu unterscheiden:

1. In α werden die beiden Zeichen B von s_i und s_j in einer Spalte angeordnet, also

$$\alpha_{s_i}[k] = \alpha_{s_j}[k] = \mathtt{B}$$

für ein k. Dann enthält das Alignment mindestens $2 \cdot |i-j|$ Gaps, da die Sequenzen um $|i-j|$ Zeichen gegeneinander verschoben werden müssen. Also in diesem Fall:

$$\begin{aligned} \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) &= \min\{\overbrace{\mathrm{sc}(\mathtt{B},\mathtt{B})}^{=0} + g \cdot \overbrace{\mathrm{sc}(\mathtt{A},\Delta)}^{=\mathrm{sc}(\Delta,\mathtt{A})=2} \mid g \geq 2 \cdot |i-j|\} \\ &= \min\{2 \cdot g \mid g \geq 2 \cdot |i-j|\} \geq 4 \end{aligned} \tag{2.10}$$

2. In α trifft ein B auf ein Δ und das andere auf ein A, also etwa

$$\alpha_{s_i}[k] = \mathtt{B},\ \alpha_{s_j}[k] = \Delta,\ \alpha_{s_j}[k'] = \mathtt{A} \text{ und } \alpha_{s_j}[k'] = \mathtt{B}$$

für $k \neq k'$ (oder umgekehrt, mit s_i und s_j vertauscht). Dann gilt:

$$\begin{aligned} \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) &= \min\{\mathrm{sc}(\mathtt{B},\Delta) + \mathrm{sc}(\mathtt{B},\mathtt{A}) + (g-1) \cdot \mathrm{sc}(\mathtt{A},\Delta) \mid g \geq 1\} \\ &= \min\{3 + 2 \cdot (g-1) \mid g \geq 1\} \geq 3 \end{aligned} \tag{2.11}$$

3. In α treffen beide Bs auf Gaps, also

$$\alpha_{s_i}[k] = \mathtt{B},\ \alpha_{s_j}[k] = \Delta,\ \alpha_{s_i}[k'] = \Delta \text{ und } \alpha_{s_j}[k'] = \mathtt{B}$$

für $k \neq k'$. Dann gilt:

$$\begin{aligned} \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) &= \min\{2 \cdot \mathrm{sc}(\mathtt{B},\Delta) + (g-2) \cdot \mathrm{sc}(\mathtt{A},\Delta) \mid g \geq 2\} \\ &= \min\{4 + (g-2) \cdot 2 \mid g \geq 2\} \geq 4 \end{aligned} \tag{2.12}$$

4. In α treffen beide Bs auf As, also

$$\alpha_{s_i}[k] = \mathtt{B},\ \alpha_{s_j}[k] = \mathtt{A},\ \alpha_{s_i}[k'] = \mathtt{A} \text{ und } \alpha_{s_j}[k'] = \mathtt{B}$$

für $k \neq k'$. Dann gilt:

$$\begin{aligned} \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) &= \min\{2 \cdot \mathrm{sc}(\mathtt{B},\mathtt{A}) + g \cdot \mathrm{sc}(\mathtt{A},\Delta) \mid g \geq 0\} \\ &= \min\{2 + g \cdot 2 \mid g \geq 0\} \geq 2 \end{aligned} \tag{2.13}$$

Damit wird offenbar das Minimum für $g = 0$ und den Fall 4 erreicht:

$$\mathrm{sc}(s_i, s_j) = \mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) \overset{(2.13)}{=} 2$$

Wegen $\mathrm{sc}(s_i, s_i) = 0$ gilt dann:

$$M_i = \sum_j \mathrm{sc}(s_j, s_i) = 2(n-1)$$

Und damit ist 1 bewiesen.

Beweis von 2:
Setzt man $\tilde{s} := \mathtt{A}^n$, so gilt $\mathrm{d}(\alpha_{s_i}, \alpha_{\tilde{s}}) = 1$ für alle i.

Wählt man $\tilde{s}$, derart dass für ein i gilt $\mathrm{d}(\alpha_{s_i}, \alpha_{\tilde{s}}) = 0$, so gilt $\tilde{s} = s_i$ und nach 1 für alle anderen Sequenzen s_j: $\mathrm{d}(\alpha_{s_j}, \alpha_{\tilde{s}}) = 2$. Das Ändern der Länge von $\tilde{s}$ oder das Einfügen von weiteren Bs in $\tilde{s}$ ergibt auch wieder eine größere Distanz. Analog zum Beweis von 1 folgt, dass $(\alpha_{s_i}, \alpha_{\tilde{s}})$ keine Gaps enthält.

Damit folgt $M_{\tilde{s}} = \sum_j \mathrm{d}(\alpha_{s_j}, \alpha_{\tilde{s}}) = n$ und mit 1:

$$\frac{M_c}{M_{\tilde{s}}} = \frac{2(n-1)}{n} > 2 - \varepsilon$$

□

Satz 2.29 *Für die Approximation von Tree Alignment mit Stern-Phylogenie durch Center-Star-Alignment läßt sich kein besserer Approximationsfaktor als* $\frac{1}{2}$ *erreichen*

Sei $\delta > 0$ beliebig vorgegeben. Dann werden wir unter Verwendung von Lemma 2.28 die Existenz einer Sequenzmenge zu einem Tree Alignment mit Stern-Phylogenie zeigen, so dass der relative Fehler größer als $\frac{1}{2} - \delta$ ist.

Offensichtlich gibt es zum gewählten $\delta > 0$ ein $\varepsilon > 0$ mit:

$$\frac{1}{2-\varepsilon} < \frac{1}{2} + \delta \tag{2.14}$$

Zu diesem ε konstruieren wir mit Lemma 2.28 die entsprechenden Sequenzen, so dass gilt: $M_c/M_{\tilde{s}} > 2 - \varepsilon$. Dann gilt für den relativen Fehler mit den Bezeichnungen aus Definition 1.16:

$$\begin{aligned}
\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} &= \frac{|M_c - M_{\tilde{s}}|}{\max\{M_{\tilde{s}}, M_c\}} = \frac{M_c - M_{\tilde{s}}}{M_c} \\
&> \frac{M_c - \frac{M_c}{2-\varepsilon}}{M_c} = 1 - \frac{1}{2-\varepsilon} \\
&\overset{(2.14)}{>} 1 - \frac{1}{2} - \delta = \frac{1}{2} - \delta
\end{aligned}$$

□

2.4 Anmerkungen

Die **NP**-Vollständigkeit von **SIMPLE MAX-CUT-B** für $B \geq 3$ ist in [Yan78] nachgewiesen.

Der Abschnitt 2.1.1 ist eine Darstellung des Beweises in [Jus99]. Dabei wurden genaue Definitionen der verwendeten Probleme ergänzt, und der Beweis gestrafft, um die Übersichtlichkeit zu verbessern. Beim Begriff der generischen Score-Matrix (vergleiche Definition 2.2) wird bei [Jus99] $x := \mathrm{sc}(\Delta, \Delta)$ gesetzt

und es muss nur $x \geq 0$ erfüllt sein. Um Konsistenz mit der Definition von SPScore in Abschnitt 1.1.2 zu wahren, wird hier diese leicht abgewandelte Version mit $x = 0$ verwendet. Auch nur bei [Jus99] zu finden ist der Beweis von Lemma 2.7.

Im Abschnitt 2.1.2 sind einige Ergebnisse zur **NP**-Vollständigkeit verschiedener Tree Alignment-Varianten aufgelistet. Weitere Ergebnisse sind im Abschnitt 2.2.2 zu finden, da alle **MAXSNP**-vollständigen Probleme auch **NP**-vollständig sind. Zum Beweis der hier angegebenen Resultate: Die **NP**-Vollständigkeit von **Tree Alignment** (ohne weitere Vorgaben) ist bei [SW92] zu finden, die von **Tree Alignment mit vorgegebenem binären Baum als Phylogenie** in [WJ94].

Für den Abschnitt 2.2.1 lieferte der Beweis von Theorem 2 in [Jus99] als Ideen die Codierung von Kanten in Sequenzen und damit verbundene Teile des Scoring-Systems. Darauf aufbauend wurde hier ein neuer, einfacherer Beweis entwickelt. Dabei wurde das Scoring-System derart modifiziert, dass auf eine Wiederholung der Kantencodierung in den Sequenzen verzichtet werden konnte. Das ermöglicht ein Wegfallen der bei [Jus99] verwendeten Skalierung mit der Funktion k^5 und damit die Verwendung einer L-Reduktion. Dadurch ist es möglich, ohne den Begriff der *gap-preserving reduction* auszukommen, deren Verhältnis zur L-Reduktion noch nicht endgültig geklärt ist und die zu einer anderen Definition von **MAXSNP** führt.

In [PY91] ist nachgewiesen, dass es Werte für B gibt, so dass das im Satz 2.16 verwendete Problem **VERTEX-COVER-B** in **MAXSNP-hard** liegt. Die Beweise von Satz 2.16 und 2.19 sind in [WJ94] vorgestellt.

Der in Abschnitt 2.3.1 vorgestellte Beweis ist eine Ausarbeitung des Abschnittes 14.6.2 „A bounded-error approximation method for SP alignment“ von [Gus97]. Hinzugefügt wurden Satz 2.24 und 2.25 um eine Verbindung mit der Notation von Kapitel 1.2.2 herzustellen. Damit liegen die Approximationsfaktoren in den beiden häufig verwendeten Versionen vor: Als relative Fehler und als Quotient von Näherungslösung und Optimum. Dies gewährleistet einen direkten Vergleich mit anderen in der Literatur angegebenen Werten.

Weiter bleibt zu erwähnen, dass es inzwischen schon Verfahren gibt, die statt dem Quotienten $2 - \frac{2}{k}$ für beliebiges, positives l, den Quotienten $2 - \frac{l}{k}$ liefern ([BLP94]).

Mit der Darstellung des Algorithmus 2.1 und der Laufzeitbetrachtung wurde hier auch eine genauere Dokumentation des Approximationsverfahrens durchgeführt als bei [Gus97].

Dem Abschnitt 2.3.2 liegt als Idee die offensichtliche Ähnlichkeit zwischen dem Tree Alignment mit Stern-Phylogenie und dem Center-Star-Alignment zugrunde, die hier erstmals diskutiert wird.

Für **Tree Alignment mit einem binären Baum als Phylogenie** wurden in [WG96] und [WJG00] Polynomialzeit-Algorithmen vorgestellt, deren Approximationsfehler und Laufzeiten von der Sequenzanzahl und -länge sowie der Tiefe des Baumes abhängen, und zusätzlich noch von einem Parameter der die Größe der in der Berechnung betrachteter Bereiche im Baum angibt. Im allgemeinen ergeben sich hier für die Praxis unbrauchbare Laufzeiten und Fehler.

Kapitel 3

Empirischer Vergleich von Alignment-Verfahren

Im Rahmen dieser Arbeit wurde ein empirischer Vergleich von Multiple-Sequence-Alignment-Verfahren unter beispielhafter Auswahl von Verfahren aus verschiedenen Gruppen durchgeführt. Als Quelle für die Testdatensätze diente eine Datenbank von Alignments, die zum Test von Alignmentverfahren erstellt wurde. Zur Bewertung wurden verschiedene Funktionen verwendet, um die Verfahren mit unterschiedlichen Kriterien zu untersuchen.

Im Folgenden werden erst die Testdaten, dann die Verfahren und danach die Bewertungsfunktionen vorgestellt. Am Ende des Kapitels erfolgt eine Auswertung der Ergebnisse.

3.1 Testdaten: BAliBASE

BAliBASE ist eine Datenbank mit Alignments, die speziell zum Vergleich von Multiple-Alignment-Verfahren erstellt wurde, und enthält verschiedene Kategorien, die die Untersuchung des Verhaltens der Verfahren bei bestimmten Eigenschaften von Alignments ermöglicht.

3.1.1 Herkunft und Erstellung der Referenzalignments

Da die Verfahren auf Eigenschaften von realen Alignments und Sequenzen beruhen ist es interessant, zu untersuchen, ob die Verfahren ohne die Eingabe von Zusatzinformationen wie etwa die Sekundärstruktur der Sequenzen zu biologisch sinnvollen Ergebnissen kommen. Dazu bietet es sich an, Testdaten zu verwenden, die realitätsnah sind und bei denen auch die biologischen Eigenschaften bekannt sind.

Deshalb wurde BAliBASE aus Sequenzen zusammengestellt, deren Verwandtschaft und (auch gemeinsame) Sekundärstrukturen bekannt sind. Als

Quelle für die Alignments dienten die Strukturdatenbanken FSSP und HOMSTRAD. Darüber hinaus wurden falls vorhanden weitere Strukturinformationen aus der HSSP Datenbank und dem VAST Webserver verwendet. Mit der PDBsum Datenbank erfolgte eine Identifizierung funktionaler Bereiche der Sequenzen und darauf aufbauend eine manuelle Verbesserung der Alignments von den Autoren von BAliBASE. Des weiteren wurde das Proteinstruktur-Vergleichsprogramm SAP verwendet, um die Alignments zu überprüfen.

Die vorliegende Beschreibung der Autoren von BAliBASE bezieht sich auf Version 1 in BAliBASE, in der es 5 verschiedene Referenz-Gruppen von Testdaten gibt. In der Gruppe 1 befinden sich die wie oben beschrieben erstellten Alignments, die sich in modifizierter Form in den Gruppen 2 bis 5 wiederfinden. Damit ist es möglich, die Auswirkungen der Änderungen und damit Hinzunahme weiterer Eigenschaften von Alignments auf Ergebnisse der Programme zu untersuchen. In diesen Gruppen befinden sich 142 Referenz-Alignments mit insgesamt mehr als 1000 Sequenzen. 58% der Zeichen in den Sequenzen befinden sich in den sogenannten „Core Blocks“, den konservierten Bereichen in den Sequenzen, für die ein gutes Alignment existiert. Die restlichen 42% befinden sich in Regionen der Sequenzen, für die kein gutes Alignment bestimmt werden kann.

Zu jedem Referenzalignment gibt es eine Übersichtsseite, auf der das Alignment mit den Sekundärstrukturdaten dargestellt ist und sich Informationen über die Zugehörigkeit der Sequenzen zu bekannten Proteinfamilien befinden. Dies ist vor allem für die Untersuchung der Frage interessant, ob sich diese auch im zu bewertenden Multiple Alignment widerspiegeln. Die Sekundärstrukturinformationen sind auch als „Feature-Table-Files“ in Programm-lesbarer Form vorhanden.

In BAliBASE 2 gibt es die zusätzlichen Gruppen 6 bis 8, die weitere Probleme behandeln, allerdings ohne Proteinfamilien- oder Sekundärstrukturinformationen in der oben angegebenen Form.

3.1.2 Referenz-Gruppen von BAliBASE

Die Eigenschaften der Alignments in den Gruppen sind folgende:

1. Diese Gruppe enthält Alignments weniger Sequenzen ähnlicher Länge, die alle evolutionär gleich weit voneinander entfernt sind. Die Gruppe ist nochmals unterteilt in drei verschiedene Längenklassen: Kurz (bis ca. 150 Zeichen), mittel (bis ca. 300) und lang. Jede dieser Klassen ist nach aufsteigender Ähnlichkeit der Sequenzen gruppiert.

2. Bei dieser Gruppe werden Gruppen eng verwandten Sequenzen (mit mehr als 25% Identität) um bis zu drei Waisensequenzen erweitert, die zwar entfernt verwandt mit den anderen Sequenzen sind und auch gemeinsame Bereiche haben, aber insgesamt weniger als 20% Identität mit den anderen aufweisen. Diese Gruppe liefert Hinweise darauf, wie sich die Anwesenheit von weit entfernt verwandten Sequenzen auf die Qualität des Gesamtalignments und auf die Qualität des Alignments der anderen mit den Wai-

sensequenzen auswirkt. Diese Gruppe ist wie innerhalb der Klassen von Gruppe 1 nach aufsteigender Länge der Sequenzen gruppiert.

3. In dieser Gruppe bestehen die Alignments aus bis zu vier Proteinfamilien, zwischen denen die Sequenzen eine Identität von weniger als 25% haben, während innerhalb der Familien die Identität über diesem Wert liegt. Damit soll untersucht werden, wie ein Verfahren derart unterschiedliche Sequenzen in ein Alignment zusammenfügen kann und wie dabei die Struktur in den Familien berücksichtigt wird. Auch hier sind die Alignments nach aufsteigender Sequenzlänge gruppiert.

4. In dieser Gruppe erfolgen bei einer bzw. wenigen Sequenzen Erweiterungen an den Enden, es kommen also auch einzelne Sequenzen mit wesentlich größerer Länge vor.

5. Diese Gruppe ist ähnlich wie Gruppe 4, nur dass diesmal die Erweiterungen in den veränderten Sequenzen als Insertionen über die Sequenzen verteilt sind und nicht nur am Rand auftreten.

6. Gruppe 6 umfasst Alignments mit vielen Wiederholungen in den Sequenzen und soll Antworten auf die Frage liefern, inwieweit Alignmentverfahren die wiederholten Bereiche den richtigen, d.h. biologisch mit der gleichen Funktion versehenen Bereichen (den Domänen) zuordnen. Neben den Alignments, die alle Sequenzen enthalten, wurden auch noch Teile von diesen als Testdaten zusammengestellt und in verschiedene Klassen eingeteilt:

   - 1(a) Gleiche Anzahl Wiederholungen einer Art
   - 1(b) Unterschiedliche Anzahl Wiederholungen einer Art
   - 2(a) Gleiche Anzahl von Wiederholungen verschiedener Arten in der gleichen Reihenfolge
   - 2(b) Gleiche Anzahl von Wiederholungen verschiedener Arten in unterschiedlicher Reihenfolge
   - 2(c) Unterschiedliche Anzahl von Wiederholungen verschiedener Arten
   - 3 Eine weitere, nicht wiederholte Domäne befindet sich in den Sequenzen
   - 4 Verschiedene Wiederholungsarten befinden sich in den Sequenzen

7. Gruppe 7 enthält Alignments transmembraner Proteine. Diese Proteine enthalten Helices, die durch die Membran von Zellen hindurchreichen und deshalb besonders wichtig für die Funktion sind.

8. Diese Gruppe enthält Sequenzen mit Inversionen, also Abschnitte deren Zeichenfolge umgedreht wurde und Permutationen der Domainabfolge, also eine Umstellung von konservierten Bereichen.

3.2 Betrachtete Verfahren

Aus den verfügbaren Verfahren wurden verschiedene beispielhaft ausgewählt, wobei darauf geachtet wurde, dass der zugrundeliegende Algorithmus gut und nachvollziehbar dokumentiert war und die Programme frei verfügbar sind. In den Vergleich wurden die häufig benutzten Standard-Verfahren CLUSTAL-W und DIALIGN einbezogen, genauso wie MSA als Implementierung des optimalen Verfahren mit SPScore. T-Coffee wurde als vielversprechende Weiterentwicklung von CLUSTAL-W mit in den Vergleich aufgenommen und HMMER, um die Qualität eines in anderen Bereichen der Bioinformatik etablierten Verfahrens zu untersuchen. Diese kleine Auswahl erhebt keinen Anspruch auf Vollständigkeit. Die verwendeten Verfahren sind in dieser Arbeit jedoch so genau dokumentiert, dass die Funktionsweise deutlich wird und damit eine bessere Begründung der Resultate ermöglicht. Trotz der kleinen Auswahl werden verschiedene Herangehensweisen an das Multiple Alignment behandelt: Optimales Verfahren für SP-Score (MSA), iteratives bzw. progressives Alignment (CLUSTAL-W, T-Coffee), lokales Alignment (DIALIGN) und ein stochastisches auf Hidden-Markov-Models basierendes Verfahren (HMMER).

3.2.1 MSA

Das Programm MSA berechnet ein bezüglich SPScore und PAM250-Scoringsystem (vergleiche Abschnitt 3.3.2) optimales Alignment. Dabei wird das von Carillo und Lipman vorgeschlagene dynamische Programmieren mit Relevanztest verwendet (Vergleiche dazu Seite 8 im Abschnitt 1.1.2.) Da dabei nicht alle Zellen der k-dimensionalen Berechnungsmatrix, sondern im Idealfall nur wenige für die Berechnung eines optimalen Alignments verwendet werden, wird diese als Edit-Graph des Alignments repräsentiert und gespeichert (vergleiche Kap. 1.1.1, Seite 4). Ein optimales Alignment entspricht hier wieder einem Weg durch den Graphen mit minimalem Score. Dabei wird ein ein gewichteter SPScore verwendet wird, dessen Gewichte aus einer durch Neighbor Joining erzeugten Näherungslösung für den phylogenetischen Baum der Sequenzen kommen, wie in [ACL89] erklärt ist. Damit soll vermieden werden, dass viele Sequenzen mit hoher Identität das Gesamtalignment dominieren und sich damit negativ auf das Alignment mit den anderen Sequenzen auswirken. Endständige Gaps werden bei MSA nicht bestraft und es wird eine affin-lineare Gap-Penalty verwendet.

Doch nun zu einer groben Darstellung des Algorithmus, wie er MSA zugrundeliegt. Auf die Erweiterung für affin-lineare Gap-Penalty wird jedoch nicht eingegangen.

Wie im Kapitel 1 schon angedeutet, wird für den Relevanztest eine untere Schranke benötigt, nämlich der Score, den ein Alignment höchstens haben darf, damit es in der Berechnung berücksichtigt wird. Zunächst wird dazu eine untere Schranke bestimmt, in die die Bewertung der paarweisen Alignments aller Sequenzen eingehen.

Seien $s_1, \ldots, s_k$ die zu alignenden Sequenzen und sei mit $g(s_i, s_j)$ die paarweise Gewichtung des Sequenzpaares (s_i, s_j) bezeichnet. $\mathrm{sc}(s_i, s_j)$ bezeichne wieder

den Score eines optimalen paarweisen Alignments von s_i und s_j. Dann setze:

$$L := \sum_{i<j} \mathrm{sc}(s_i, s_j) g(s_i, s_j) \tag{3.1}$$

Damit wird eine grobe untere Schranke für die betrachteten Alignments abgeschätzt. Mit dem Center-Star-Verfahren (vergleiche Abschnitt 2.3.1) wird ein Alignment α^c bestimmt, nun aber nicht direkt dessen Score als Schranke verwendet, sondern die Schranke wie folgt geschätzt:

$$U := L + \sum_{i<j} (\min(\varepsilon, |\mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_j}) - \mathrm{sc}(s_i, s_j)|) g(s_i, s_j)) \tag{3.2}$$

Die Bezeichnung $\mathrm{d}(\alpha^c_{s_i}, \alpha^c_{s_j})$ hat dabei die in schon Abschnitt 2.3.1 definierte Bedeutung und gibt den Abstand zwischen den Sequenzen $\alpha^c_{s_i}$ und $\alpha^c_{s_j}$ an, während ε eine vorgegebene Konstante ist. Durch diese Abschätzung kann eine zu niedrige Schranke für den Relevanztest entstehen, die dann dazu führt, dass MSA gar kein Alignment bestimmt.

Mit dieser Schranke wird nun direkt für jeden betrachteten Knoten im Edit Graph bestimmt, ob er zu einem Weg eines Alignment α mit Score $\leq U$ gehört. Dazu werden alle Projektionen auf zwei Sequenzen an diesem Knoten betrachtet und folgende Ungleichung gibt dann an, ob der Knoten auf einem entsprechenden Weg liegen kann:

$$\begin{aligned} U - L &\geq \sum_{i<j} g(s_i, s_j)(\mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) - \mathrm{sc}(s_i, s_j)) \\ &\geq g(s_p, s_q)(\mathrm{d}(\alpha_{s_p}, \alpha_{s_q}) - \mathrm{sc}(s_p, s_q)) \\ &\Rightarrow g(s_i, s_j)\mathrm{d}(\alpha_{s_i}, \alpha_{s_j}) \leq g(s_i, s_j)\mathrm{sc}(s_i, s_j) + U - L \end{aligned} \tag{3.3}$$

Neben dem Relevanztest wird nun noch eine Methode benötigt, um einen optimalen Weg im Edit-Graphen zu finden, also den Weg eines optimalen Alignments. Dazu wird der Dijkstra-Algorithmus verwendet, der im Edit-Graphen die im Algorithmus 3.1 beschriebenen Operation ausführt, wobei das Gewicht $w(v)$ jedem Knoten v den Score eines optimalen Alignments mit Weg vom Startknoten bis zu v zuordnet.

Für jeden Nachbarn v' wird also überprüft, ob v zu einem günstigeren Weg zu v führt. Wenn ja, wird dieser günstigere Weg v' zugewiesen. In der Prioritätswarteschlange werden demnach alle Knoten gehalten, die schon als mögliche Fortführung eines Teilalignments betrachtet wurden. Die Kantengewichte $w((v, v'))$ hängen hier von den beim Übergang von Knoten v zu v' alignten Zeichen ab, also von den Matches, Mismatches und Gaps an dieser Stelle. Prinzipiell entspricht diese Vorgehensweise immer noch dem Needleman-Wunsch-Algorithmus.

Um diesen Algorithmus mit dem Referenztest zu kombinieren, wird einfach bei jedem Knoten, der aus der Prioritätswarteschlange genommen wird überprüft, ob dieser relevant ist. Die dafür benötigten Werte $\mathrm{sc}(\alpha_{s_i}, \alpha_{s_j})$ können

Algorithmus 3.1 Dijkstra-Schritt

Nehme den Knoten v mit der höchsten Priorität aus der Prioritätswarteschlange und überprüfe für jeden Nachbarn v', der von v aus erreicht werden kann, ob $w(v')$ schon ein Wert zugewiesen wurde (also sich dieser in der Prioritätswarteschlange befindet).

- Falls nicht, setze: $w(v') := w(v) + w((v, v'))$
- sonst prüfe, ob $w(v') > w(v) + w((v, v'))$
  Wenn dies der Fall ist, setze: $w(v') := w(v) + w((v, v'))$

Dann nehme v' in die Prioritätswarteschlange auf.

mit paarweisen Alignments effizient vorher berechnet werden. Ein zu einem gefundenen Weg passendes Alignment kann am Ende ausgegeben werden, indem der Weg bis zum Startpunkt zurückverfolgt wird. Dazu ist es allerdings nötig, während der Berechnung des optimalen Weges zu speichern, welche Kanten im Algorithmus 3.1 ausgewählt wurden.

Der größte Vorteil von MSA, nämlich dass es ein optimales Alignment bestimmt, führt aber auch gleich zum größten Nachteil, der zu hohen Laufzeit. Je nach Größe der Schranke U können auch sehr viele oder zu wenige Knoten betrachtet werden. So führt bei größeren Eingabedatenmengen eine hohe Schranke zu sehr großem Speicherplatzbedarf und Zeitaufwand oder es wird bei zu niedriger Schranke gar kein Alignment bestimmt. Für den Vergleich wurden verschiedene Konstanten von MSA vergrößert, um mehr Alignments betrachten zu können, als ursprünglich vorgesehen. Im Einzelnen sind dies die maximale Anzahl der Sequenzen und die maximale Länge. Im Vergleich in dieser Arbeit kam Version 2.0 von MSA zum Einsatz.

3.2.2 CLUSTAL W

CLUSTAL W gehört zu den progressiven Alignment-Verfahren, was bedeutet, dass verschiedene Heuristiken kombiniert werden. Nachdem als eine Näherungslösung für den phylogenetischen Baum der sogenannte „Guide Tree“ bestimmt wurde, wird an diesem durch paarweise Alignments das Alignment aufgebaut, wobei Parameter wie Scoring-System oder Gap-Penalty angepasst werden.

Zunächst werden die Scores aller paarweisen Alignments bestimmt, wobei entweder dynamisches Programmieren zum Einsatz kommt, oder die Scores approximiert werden. Diese werden als Gewichte in der folgenden Form gespeichert:

$$g(s_i, s_j) = 1 - [\text{Identität von } s_i \text{ und } s_j] \tag{3.4}$$

Dann wird mit dem Neighbor-Joining-Verfahren basierend auf diesen paarweisen Abständen ein Guide Tree bestimmt, der eine Näherungslösung für den tatsächlichen phylogenetischen Baum der Sequenzen darstellt. Dazu wird das

Sequenzpaar mit dem geringsten Abstand zu einem Knoten zusammengefasst, eine entsprechend modifizierte Distanzmatrix bestimmt und mit dieser dann genauso verfahren, und dies solange wiederholt, bis man alle Sequenzen in einem Baum angeordnet hat. Erst am Ende wird noch von diesem Baum ein bestimmter Knoten als Wurzel ausgezeichnet. Ein Beispiel für einen solchen Baum ist der erste Baum in Abbildung 3.1, wobei natürlich der Guide-Tree selten wie dort ein vollständiger Baum ist.

An diesem Baum wird nun sukzessive das Alignment aufgebaut, und zwar ausgehend von den Blättern zur Wurzel, wie auch in Abbildung 3.1 zu verfolgen ist. Die zwei durch einen inneren Knoten u verbundene Blätter v_1 und v_2 mit dem geringsten Abstand werden ausgewählt und ein optimales paarweises Alignment α der Sequenzen an diesen beiden Knoten bestimmt. Dann wird der Teilbaum aus den Knoten v_1, v_2 und u durch einen neuen Knoten ersetzt, der als Markierung das Alignment α erhält. Mit dem neuen Baum wird die diese Operation solange wiederholt, bis zum Schluß nur noch die Wurzel und damit das Gesamtalignment übrig ist. Sind in einem Schritt Teil-Alignments beteiligt, die zu einem Alignment zusammengefasst werden sollen, so werden diese Teil-Alignments α_{v_1} und α_{v_2} jeweils wie eine Sequenz behandelt und bei der Scoreberechnung in jeder Spalte der Mittelwert der Scores zwischen allen Sequenzen in dieser Spalte genommen. Bei 4 Sequenzen in α_{v_1} und 2 Sequenzen in α_{v_2} würde so in jeder Spalte der Mittelwert der $2 \cdot 4 = 8$ Vergleiche als Spaltenscore verwendet. Da die Teilalignments in späteren Schritten wie eine Sequenz behandelt werden, bleiben Gaps, die einmal eingefügt wurden, bis zum Schluß erhalten - was natürlich zu Fehlern führt. Zusätzlich werden die Gewichtungsfaktoren aus (3.4) verwendet, um einen gewichteten Score zu erhalten.

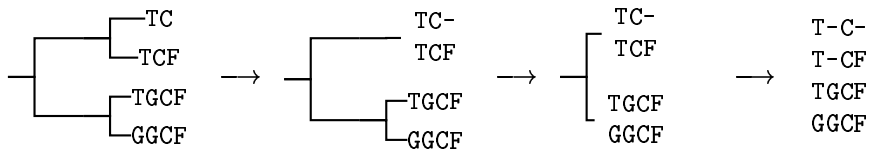


Abbildung 3.1: Beispiel: Progressives Alignment

Die paarweisen Distanzen werden aber nicht nur zur Berechnung des Guide-Trees und zur Gewichtung verwendet, sondern auch um die Bewertung an die jeweils betrachteten Sequenzen anzupassen. Da CLUSTAL-W mit den PAM oder BLOSUM Scoringsystemen arbeitet (vergleiche dazu Abschnitt 3.3.2), steht jeweils eine ganze Reihe von Systemen zur Verfügung, die immer eine bestimmte evolutionäre Distanz beschreiben. Unter diesen wird eine passende ausgewählt, wobei eine größere Ähnlichkeit der Sequenzen dann zur Wahl eines Systems für kurze evolutionäre Zeiträume führt.

Neben der Wahl des Scoringsystems wird auch die Gap-Penalty angepasst. Die Anfangswerte für die Aufstellung der Distanzmatrix und der ersten Alignments ergeben sich wie folgt: Die vorgegebene Gap-Open-Penalty wird für zwei Sequenzen mit den Längen n_1 und n_2 an die Sequenzlängen angepasst und

mit dem durchschnittlichen Mismatch-Wert m des verwendeten Scoring-Systems und der Ähnlichkeit der Sequenzen id skaliert:

$$\text{GOP} := (\text{GOP} + \log(\min\{n_1, n_2\})) \cdot m \cdot id$$

Die Gap-Extension-Penalty wird abhängig vom Unterschied zwischen den Sequenzlängen n_1 und n_2 verändert:

$$\text{GEP} := \text{GEP}(1 + |log(n_1/n_2)|)$$

Im Zuge des progressiven Alignments erhält dann jede Position in den Sequenzen eine individuelle Gap-Open-Penalty, die von

- der Anzahl der Sequenzen ohne Gaps in den Teilalignments
- der Entfernung zu bestehenden Gaps
- den chemischen Eigenschaften der Aminosäurereste

abhängt, was etwa zur Vermeidung vieler kleiner Gaps nebeneinander führen soll. In dieser Arbeit wurde Version 1.8 von CLUSTAL-W verwendet, das Teil von CLUSTAL-X ist, einer Erweiterung von CLUSTAL-W um eine graphische Oberfläche.

3.2.3 T-Coffee

Eine Erweiterung von CLUSTAL-W ist das T-Coffee-Verfahren. Der Grobablauf ist dem vom CLUSTAL-W sehr ähnlich, nur dass zunächst eine Bibliothek von paarweisen Alignments aufgebaut wird und unter Verwendung dieser Zusatzinformationen jedes Alignment während des progressiven Verfahrens mit einem individuellen Scoringsystem erstellt wird.

Zunächst werden zwei Alignment-Bibliotheken aufgebaut: Die eine enthält alle paarweisen Alignments, erstellt mit CLUSTAL-W. Die andere enthält lokale Alignments zwischen je zwei Sequenzen. Dazu werden mit dem Programm Lalign aus dem FASTA-Paket alle lokalen Alignments bestimmt und die zehn besten sich nicht überlappenden ausgewählt. In diesen beiden Bibliotheken stehen dann die Zuordnungen der einzelnen Zeichen durch die Alignments und als Gewichtungen davon die Scores der Alignments. In dieser Bibliothek stehen demnach Einträge der Form:

3. Zeichen, Sequenz 23 (A) + 17. Zeichen, Sequenz 4 (A) - Gewicht 88

Diese beiden Bibliotheken werden nun zu einer zusammengefasst, wobei bei in beiden vorhandenen Zuordnungen die Summe der Einzelgewichte als Gewicht genommen wird. Diese Bibliothek wird jetzt noch erweitert, indem für jedes Paar von Sequenzen und jede weitere Sequenz Alignments über diese dritte Sequenz erstellt werden. Ein Beispiel ist in Abbildung 3.2 zu sehen. Dort werden für die Sequenzen s_1 und s_2 zusätzlich zu deren Alignment (oberste Zeile) noch die Alignments betrachtet, die entstehen, wenn man die anderen Sequenzen,

hier s_3 und s_4, hinzunimmt. Nun werden die Gewichte der Zeichenzuordnungen von (s_1, s_2) mit den Gewichten der Zuordnungen (s_1, s_3) und (s_3, s_2) erweitert, indem neue hinzugenommen werden und bei bereits vorhandenen doppelte addiert werden. Gleiches geschieht mit Sequenz s_4. Damit ändert sich das in der Bibliothek enthaltene Alignment (s_1, s_2) auf die in der letzten Zeile angegebene Form. In Abbildung 3.2 sind die betrachteten Zuordnungen als vertikale Balken gekennzeichnet.

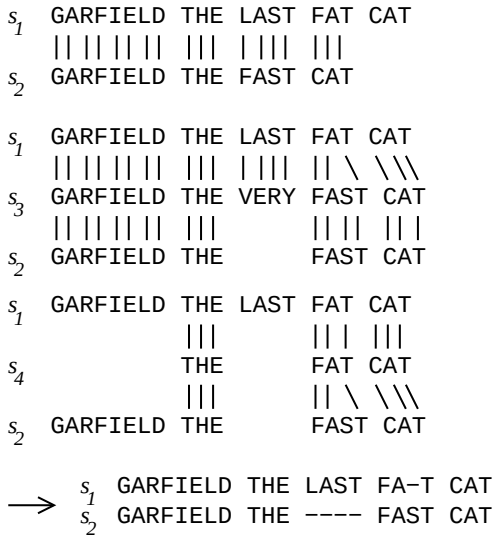


Abbildung 3.2: Beispiel: Erweiterung der T-Coffee Bibliothek. Vertikale Balken kennzeichnen die Zuordnungen von Zeichen in den Alignments, wobei sowohl Matches als auch Mismatches vorkommen.

Mit den Gewichtungen der Zeichenzuordnungen in den Alignments als Scoring-System und ohne Gap-Penalty werden nun die Schritte ausgeführt, die auch CLUSTAL-W ausführt:

1. Bestimmung aller paarweisen Distanzen
2. Bestimmung des Guide-Tree mittels Neighbor Joining
3. Progressives Alignment entlang des Guide-Trees

Ein Nachteil dieser Methode liegt auf der Hand: Durch die Speicherung aller Alignments benötigt T-Coffee sehr viel Speicherplatz, weshalb auch im Rahmen des Verfahrensvergleichs manche Testdatensätze zu groß waren und das Programm mangels Speicherplatz vom Betriebssystem beendet wurde. T-Coffee ist das einzige Programm, das direkt nach der Entwicklung mit BAliBASE getestet wurde. Dazu wurde BAliBASE Version 1 verwendet, bei der die großen Datensätze noch nicht enthalten waren und somit die genannten Probleme nicht

auftraten. Verglichen wurde T-Coffee dort mit CLUSTAL-W, DIALIGN und Prrp, einem Programm, das hier nicht untersucht wird. Interessant sind auch die Untersuchungen der Verbesserungen, die aus der Verwendung zweier Bibliotheken am Anfang und der oben beschriebenen Erweiterung der Bibliothek entstehen. Für den in dieser Arbeit vorgenommenen Vergleich wurde die Version 1.37 verwendet.

3.2.4 DIALIGN

DIALIGN ist ein Programm, das mit stark konservierten Bereichen arbeitet. Um ein Alignment zu erstellen werden bei DIALIGN nicht einzelne Zeichen, sondern ganze Segmente miteinander verglichen und aligned.

Die Alignmenterstellung bei DIALIGN bestimmt in paarweisen Alignments Bereiche, die ohne Gaps aligned werden. Diese treten im Edit-Graphen als Diagonalen auf, weshalb sie auch hier als „Diagonalen“ bezeichnet werden. Zunächst werden alle Diagonalen in allen Sequenzpaaren bestimmt, gewichtet und dann mit einem Greedy Algorithmus zu einem Multiple Alignment zusammengesetzt.

Dabei kann es vorkommen, dass verschiedene Diagonalen für die gleichen Zeichen in einer Sequenz verschiedene Zuordnungen zu Zeichen in der anderen Sequenz liefern. Um daraus die widersprüchlichen Fälle herausfiltern zu können, wird ein Konsistenzbegriff definiert: Zwei Diagonalen D_1 und D_2 zwischen denselben Sequenzen s_1 und s_2 heißen *konsistent*, wenn sich die Positionen der durch die Diagonalen zugeordneten Zeichen in beiden Sequenzen in der gleichen Reihenfolge befinden, ansonsten *inkonsistent* (siehe auch Abbildung 3.3).

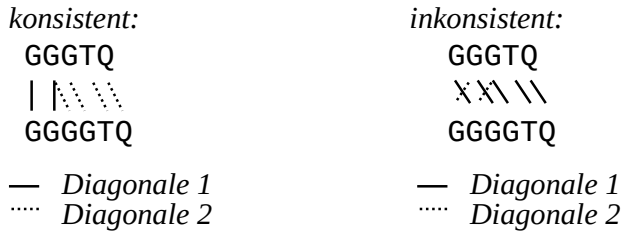


Abbildung 3.3: Beispiel: Konsistente und inkonsistente Diagonalen

Zur Bewertung wird jeder Diagonalen ein Gewicht bzw. eine Score zugeordnet. Betrachtet man eine Diagonale D der Länge l mit m Matches und nimmt man an, dass jede Aminosäure an jeder Stelle einer Sequenz mit der gleichen Wahrscheinlichkeit $p = \frac{1}{20}$ vorkommt[1], so ergibt sich für die Wahrscheinlichkeit

[1] Dies ist ein sehr stark vereinfachtes Modell, das keine Erkenntnisse über die Eigenschaften der 20 Aminosäuren berücksichtigt.

$P(l,m) = P$(Es gibt Diagonale der Länge l mit mindestens m Matches):

$$P(l,m) = \sum_{i=m}^{l} \binom{l}{i} p^i (1-p)^{l-i}$$

Mit $E(l,m) := -\ln(P(l,m))$ wird dann die Diagonale D gewichtet durch:

$$w(D) := \begin{cases} E(l,m), & \text{falls } E(l,m) > T \\ 0, & \text{sonst} \end{cases} \tag{3.5}$$

Dabei ist T ein fest vorgegebener Parameter des Verfahrens und l und m sind die Eigenschaften der Diagonalen D, wie oben. So ist $w(D)$ hoch, wenn es unwahrscheinlich ist, dass die beobachtete Diagonale D rein zufällig auftritt. Dies führt zum Score den DIALIGN für die paarweise Alignments verwendet, der aber genauso für Multiple Alignments benutzt werden kann: Der Score einer konsistenten Menge von Diagonalen $\{D_1, \ldots, D_k\}$ ist gerade $\sum_{i=1}^{k} w(D_i)$. Der Score eines optimalen Alignment ist der maximale Score von konsistenten Diagonalenmengen des Alignments.

Eine Menge von konsistenten Diagonalen mit maximalem Score für zwei Sequenzen wird bei DIALIGN mittels mit dynamischem Programmieren bestimmt. Die Diagonalen aus allen paarweisen Berechnungen werden danach nach absteigender Gewichtung sortiert in eine Liste aufgenommen und mit einem Greedy-Algorithmus zu einem Multiple Alignment zusammengestellt: Von den verbleibenden wird jeweils diejenige zu den bereits ausgewählten Diagonalen konsistente hinzugenommen, die das höchste Gewicht hat. Zuvor werden noch die Gewichte von Diagonalen, die sich mit anderen überlappen, verändert, damit etwa Bereiche, die in allen Sequenzen vorkommen bevorzugt behandelt werden.

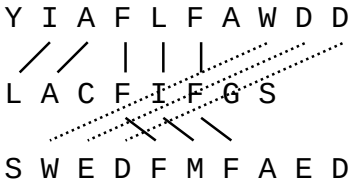
Ein Beispiel für den bisher vorgestellten Algorithmus ist als „1. Iterationsschritt“ in Abbildung 3.4 zu sehen. Werden keine konsistenten Diagonalen mehr gefunden, werden unter Berücksichtigung aller schon in das Multiple Alignment eingebauten Diagonalen weitere Diagonalen bestimmt, wie in der Abbildung unter „2. Iterationsschritt“ gezeigt. Diese Iteration wird solange fortgeführt, bis sich keine neuen konsistenten Diagonalen mehr ergeben.

DIALIGN wurde als Methode entwickelt, die gute lokale Alignments innerhalb der Multiple Alignments liefert und die Schwächen des Needleman-Wunsch-Algorithmus umgehen soll. Fraglich ist, ob ein totaler Verzicht auf ein Gap-Konzept immer zu sinnvollen Alignments führen kann. Die vorliegende Version des Programms ist 2.1.

3.2.5 HMMER

Hidden Markov Modelle (HMMs) sind stochastische Modelle, die in diesem Fall die Struktur von Sequenzklassen wiedergeben. Beim Programmpaket HMMER wird mit dem Programm `hmmt` zu den gegebenen Sequenzen ein Profile-HMM erstellt und dann mit dem Programm `hmma` ein dazu passendes Alignment bestimmt und ausgegeben.

1. Iterationsschritt



D_1 = IA / LA

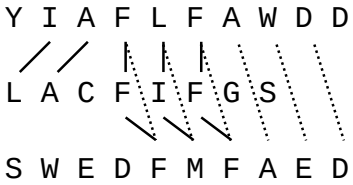
D_2 = FLF / FIF

D_3 = WDD / WED

D_4 = FIF / FMF

Diagonalengewichte	D_1	D_2	D_3	D_4
ohne Überlappung	0.2	2.6	4.7	2.2
mit Überlappung	0.2	5.3	4.7	4.9

2. Iterationsschritt



D_5 = FLFAWD / FMFAED

Abbildung 3.4: Beispiel: Iterationsschritte in DIALIGN

Mit Profile-HMMs wird versucht, den stochastischen Prozess hinter der Bildung von Protein-Sequenzen zu modellieren. Ein Profile HMM besteht aus

- einer Menge von Zuständen $\{b, e, m_1, m_2, \ldots, m_n, i_0, \ldots, i_n, d_1, \ldots, d_n\}$,
- zu jedem in Abbildung 3.5 mit Pfeilen gekennzeichneten Zustands-Übergang (q_1, q_2) einer Übergangswahrscheinlichkeit a_{q_1,q_2} und
- für alle Zustände $q \in \{m_1, m_2, \ldots, m_n, i_0, \ldots, i_n\}$ und alle Zeichen σ für Proteinsäuren der Ausgabewahrscheinlichkeit $b_{q,\sigma}$.

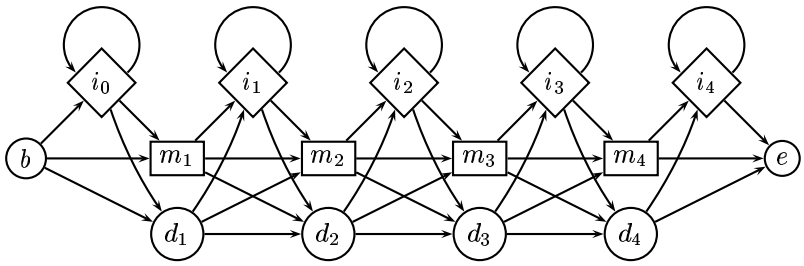


Abbildung 3.5: Ein Profile-HMM

Damit wird folgender Prozeß simuliert: Zu jedem Zeitpunkt t befindet sich das Modell in einem der Zustände; bezeichnen wir diesen mit q_t. Ist dies ein Insertions-Zustand i_j oder ein Main-Zustand m_j, so wird zufällig ein Zeichen σ_t ausgegeben, und zwar mit der Wahrscheinlichkeit b_{q_t,σ_t}. Falls sich das System noch nicht im Endzustand $q_t = e$ befindet, geht es in einen Zustand q_{t+1} über, wobei einer der zulässigen Zustände ausgewählt wird und die Wahrscheinlichkeit für den Übergang in einen bestimmten Zustand dann gerade durch $a_{q_t,q_{t+1}}$ angegeben ist. Ausserdem gilt stets: $q_0 = b$.

Eine Aminosäuresequenz entspricht den bei einem Random-Walk durch das Modell erzeugten Ausgaben. Im in Abbildung 3.5 angegebenen HMM wären also für folgende Sequenzen diese Wege möglich, wobei auch andere Wege zur gleichen Ausgabe führen können:

Sequenz	Ein Weg, bei dem die Sequenz ausgegeben wird
GGCT	b, m_1, m_2, m_3, m_4, e
ACCGAT	$b, i_0, m_1, d_2, m_3, i_3, i_3, m_4, e$
CT	b, m_1, d_2, d_3, m_4, e

Bei gegebenem Profile-HMM M ist die Wahrscheinlichkeit für einen Weg $(q_1, \ldots, q_{r'})$ mit der Ausgabe $x_1 \ldots x_r$, $P(x_1 \ldots x_r, (q_1, \ldots, q_{r'})|M)$ gerade das Produkt aller Wahrscheinlichkeiten der verwendeten Übergänge und Ausgaben. Die Wahrscheinlichkeiten für eine Ausgabe $x_1 \ldots x_r$ bei einem HMM M ist dann die Summe aller Wege, die diese Ausgabe erzeugt haben können:

$$P(x_1 \ldots x_r|M) = \sum_{q_1=b}^{e} P(x_1 \ldots x_r, (q_1, \ldots, q_{r'})|M) \tag{3.6}$$

Um ein Modell zu finden, dass die Eigenschaften einer vorgegebenen Sequenzmenge $\{s_1, \ldots, s_k\}$ am besten beschreibt, sucht man ein Profile HMM M, für das folgende Wahrscheinlichkeit maximal ist:

$$P(\{s_1, \ldots, s_k\}|M) = \prod_{j=1}^{k} P(s_j|M) \tag{3.7}$$

Dabei können jedoch nicht alle möglichen Modelle betrachtet werden, da hierfür verschiedene Modell-Längen (hier n) und zu jedem Zustand alle Möglichkeiten für Ausgabewahrscheinlichkeiten und für Übergangswahrscheinlichkeiten betrachtet werden müssten. Also kommen hier Heuristiken zum Einsatz, beispielsweise der Baum-Welch-Algorithmus 3.2.

Laut [KBM+93] ist die Konvergenz des Verfahrens sehr gut und endet meistens innerhalb von 10 Iterationen. Das einzige Problem bezüglich Laufzeit ist Schritt 2, in dem alle zulässigen Wege betrachtet werden müssen. Da systematisches Durchprobieren zu exponentiellem Aufwand führen würde, werden hier das sogenannte Forward-Backward-Verfahren oder der Viterbi-Algorithmus in einer mit dynamischem Programmieren implementierten Variante verwendet, die dann polynomiellen Zeitaufwand haben.

Algorithmus 3.2 Baum-Welch-Algorithmus

1. Es wird ein Anfangs-Modell erstellt, in das eventuell schon bekannte Strukturinformationen aufgenommen werden können.
2. Für jede Sequenz werden alle möglichen Wege bestimmt. Für jede Übergangs- und Ausgabewahrscheinlichkeit p werden entsprechend der Anzahl der Wege mit diesen Übergängen und Ausgaben Schätzer p' bestimmt.
3. Das Modell wird modifiziert, indem alle Wahrscheinlichkeiten durch die Schätzer aus Schritt 2 ersetzt werden.
4. Schritte 2 und 3 werden solange mit den jeweils geänderten Modellen durchgeführt, bis nur noch vernachlässigbar kleine Änderungen auftreten.

Im HMMER-Paket ist Algorithmus 3.2 noch derart erweitert, dass die Länge des Modells während der Iteration angepasst wird. Ein sinnvoller Anfangswert für diese Größe ist etwa die durchschnittliche Länge der Sequenzen, die durch das Modell repräsentiert werden sollen.

Hat man nun ein Profile-HMM zu einer Menge von Sequenzen bestimmt, ist es möglich, ein dazu passendes Multiple Alignment zu berechnen. Dazu wird für jede Sequenz beispielsweise mit dem Viterbi-Algorithmus ein zulässiger Weg maximaler Wahrscheinlichkeit bestimmt und entsprechend diesem die Sequenz ausgegeben: In jedem Insertions- oder Main-Zustand wird ein Zeichen der Sequenz ausgegeben, in jedem Deletions-Zustand ein Gap. Zusätzlich führen Insertions-Zustände zu Gaps in den anderen Sequenzen. Die Spalten, die aus den Main- bzw. Deletions-Zuständen entstehen dienen also zur Ausrichtung der Sequenzen beim Alignment, während die Insertionen dazwischengeschoben werden. Für die oben betrachteten Sequenzen und Wege ergibt sich dann folgendes Alignment (Abbildung 3.6):

```
A C - C G A T
. G G C . . T
. C - - . . T
```

Abbildung 3.6: Alignment der Beispielsequenzen

Dabei werden Gaps durch zwei Zeichen repräsentiert: - kennzeichnet einen Deletions-Zustand in der jeweiligen Sequenz, das Zeichen . einen Insertions-Zustand in einer anderen Sequenz. Dadurch werden im Alignment sehr viele Gaps eingefügt, die zu biologisch fragwürdigen Alignments führen.

Das HMMER Verfahren ist als stochastisches Verfahren auf eine große Menge homogener Trainingsdaten angewiesen, was beim Trainieren eines Modells nur mit den zu alignenden Sequenzen nicht der Fall ist: Dort sind zu wenige Sequen-

zen vorhanden und ausserdem unterscheiden sie sich meistens so stark, dass das Trainieren nicht zum gewünschten Erkennen von Eigenschaften durch das Modell führt. Die Funktionen von Proteinen können oft nicht durch Profile-HMMs erkannt werden, da es sich meistens um tertiär-struktur-abhängige[2] Eigenschaften handelt. 3D-Strukturen stellen Abhängigkeiten zwischen nicht benachbarten Aminosäuren eines Proteins dar, die also in HMMs durch Abhängigkeiten zwischen nicht benachbarten Zuständen modelliert werden müßten.

In der Bioinformatik gibt es noch andere Anwendungsgebiete für HMMs, wozu etwa die Strukturanalyse oder HMM-Datenbanken zählen, bei denen Sequenzen anhand ihrer Struktur klassifiziert werden und es mit Abfragen solcher Datenbanken möglich ist, neue Sequenzen bestimmten bekannten Gruppen zuzuordnen und damit Aussagen über die Funktion oder Struktur zu finden. Andere Ansätze verwenden HMMs auf DNA-Sequenzen, um Gene zu finden, also auch Exons und Introns zu unterscheiden.

In der aktuellen Version von HMMER besteht nicht die Möglichkeit, mit unalignten Sequenzen ein Profile-HMM zu trainieren. Deshalb wurde hier die ältere Version 1.84 eingesetzt.

3.3 Bewertungsfunktionen

Eine wichtige Frage beim Vergleich der Verfahren ist die nach der Bewertungsfunktion. Statt die Bewertungsfunktionen der einzelnen Verfahren für einen Vergleich heranzuziehen, werden verschiedene Bewertungsfunktionen verwendet, die keinem der Verfahren exakt entsprechen. In der BAliBASE-Untersuchung [TPP99b] wurde ein Differenz-Score eingesetzt, der einen direkten Vergleich mit den Referenz-Alignments gestattet. Weiter wurde hier eine traditionelle Scoring-System-Variante gewählt, da dies eine übliche Formalisierung der Qualität von Alignments ist und nicht die Referenz-Alignments als einzige optimale Lösung betrachtet. Aus dieser Klasse von Bewertungsfunktionen wurde das BLOSUM62-Scoring-System verwendet. Zusätzlich wurden diese beiden Scores noch um die Berücksichtigung von Sekundärstruktur und Proteinfamilien erweitert. In diesem Abschnitt werden sie alle kurz vorgestellt.

3.3.1 Differenz-Score

Um einen direkten Vergleich mit den Referenz-Alignments in der BAliBASE-Datenbank zu ermöglichen, kann man einen Differenz-Score verwenden, der die Unterschiede des untersuchten Alignments zum Referenzalignment beschreibt. Dieser Score ordnet natürlich den Referenzalignments eine sehr große Bedeutung zu, da sie als das einzige Optimum betrachtet werden. Abweichungen sind durch Gewichtungen möglich, wie sie etwa in den Abschnitten 3.3.3 und 3.3.4 vorgestellt werden. Beim Verfahrensvergleich im Rahmen der Veröffentlichung von BAliBASE wurde dieser Score normiert. Darauf wird hier verzichtet, da-

[2] Zu den verschiedenen 3D-Strukturen vergleiche Abschnitt 3.3.3.

mit eine einfachere Anwendung der später vorgestellten Gewichtungen und der damit verbundenen Auswertungen möglich ist.

Der Differenz-Score vergleicht die Positionen von Zeichen im Referenzalignment mit denen im zu bewertenden. Daher bietet es sich an, eine Funktion zu definieren, die jeder Position eines Zeichens in einer Sequenz s die Position im Alignment α_s zuordnet. Sei s eine Sequenz und α_s die dazugehörige Sequenz im Alignment α. Dann sei für für jedes Zeichen $s[i]$ die Position dieses Zeichens in α_s mit $\pi_\alpha(s[i])$ bezeichnet. Sofern es sich nicht um einen Gap handelt, liefert dann die Umkehrfunktion $\pi_\alpha^{-1}(\alpha_{s_i}[j])$ die Position des Zeichens $\alpha_{s_i}[j]$ in der Sequenz s_i.

Zu den Sequenzen $s_1, \ldots, s_k$ sei im folgenden mit α^R das Referenzalignment und mit α das zu bewertende bezeichnet. Dann ist der Scorebeitrag $\mathrm{sc}(\alpha_{s_i}[j], \alpha_{s_{i'}}[j'])$ der zwei Zeichen in den Sequenzen α_{s_i} bzw. $\alpha_{s_{i'}}$ definiert als:

$$\mathrm{sc}(\alpha_{s_i}[j], \alpha_{s_{i'}}[j']) = \begin{cases} 0 & \text{falls } \pi_{\alpha^R}(\pi_\alpha^{-1}(\alpha_{s_i}[j])) = \pi_{\alpha^R}(\pi_\alpha^{-1}(\alpha_{s_{i'}}[j'])) \\ 1 & \text{sonst} \end{cases} \tag{3.8}$$

Die beiden Zeichen führen also zu einem Scorebeitrag von 0, wenn sie im Referenzalignment in der gleichen Spalte stehen, also aligned werden, ansonsten zu 1. Der Gesamtscore ergibt sich wieder als SPScore dieser paarweisen Scores:

$$\mathrm{sc}(\alpha) = \sum_{j=1}^{|\alpha_{s_1}|} \sum_{i<i'} \mathrm{sc}(\alpha_{s_i}[j], \alpha_{s_{i'}}[j]) \tag{3.9}$$

Dadurch kann genauso wie bei den anderen SPScores eine gewichtete Version durch Hinzunahme von Gewichtungsfaktoren vor den paarweisen Scores erzeugt werden. Im ungewichteten Fall (3.9) erhält man so die Anzahl der im zu bewertenden Alignment anders als im Referenzalignment alignten Zeichenpaare.

3.3.2 Scores mit Austauschmatrizen

In Kapitel 1 wurde der Begriff des Scoring-Systems allgemein eingeführt (vgl. Definition 1.5). Diese können sowohl für paarweise als auch Multiple Alignments eingesetzt werden, um diese zu bewerten und damit die Ähnlichkeit von Sequenzen zu bestimmen. Da jedem Austausch von zwei Aminosäuren ein Score zugeordnet wird, ist es möglich, unterschiedlich wahrscheinliche Substitutionen unterschiedlich zu bewerten. So wird ein Austausch zwischen chemisch völlig verschiedenen Aminosäuren zu einem sehr niedrigen Score führen, während eine nicht veränderte Aminosäure (also keine Substitution) zu einem hohen Score führt.

Die beiden bekanntesten Vertreter von Scoring-Matrizen, die auf empirischen Auswertungen von biologischen Eigenschaften beruhen, sind die PAM und BLOSUM-Substitutionsmatrizen. Beide stellen jeweils eine ganze Reihe von Substitutionsmatrizen dar, die jeweils die Mutationen in einer Zeitspanne bei einer festen Mutationsrate beschreiben.

Dayhoff- oder PAM-Substitutionsmatrizen

Die PAM-Matrizen, die auch als Dayhoff-Matrizen bekannt sind, spiegeln Punkt-Mutationen wider (daher auch der Name: PAM - *Accepted Point Mutations*), also Mutationen einer einzigen Aminosäure. Dazu wurden Sequenzen von 71 Gruppen eng verwandter Proteine betrachtet und dort die Häufigkeiten der Mutationen von Aminosäuren im phylogenetischen Baum jeder Gruppe bestimmt. Diese Häufigkeit der Mutation von Aminosäure σ_i zu σ_j wird dann mit n_{ij} bezeichnet. Weiter wurde $n_{ij} = n_{ji}$ angenommen und noch bestimmt, wie oft eine Aminosäure sich in einem Zeitschritt insgesamt verändert, bezeichnet mit m_i (der „Mutability"). Mit einem Skalierungsfaktor λ ergibt sich dann für die Wahrscheinlichkeit, dass in einer Zeiteinheit Aminosäure σ_i zu σ_j mutiert, als:

$$M_{ij} = \frac{\lambda m_i n_{ij}}{\sum_i n_{ij}} \tag{3.10}$$

Und für die Wahrscheinlichkeit, dass keine Mutation der Aminosäure σ_j stattfindet:

$$M_{jj} = 1 - \lambda m_j \tag{3.11}$$

Betrachtet man einen größeren Zeitraum von t Zeiteinheiten, so ergibt sich als Übergangsmatrix dafür als die wiederholte Anwendung der Matrix M:

$$M^t = \underbrace{M \cdots M}_{t\text{-mal}}$$

Zur Bewertung wird allerdings nicht eine dieser Matrizen M^t verwendet, was zu einer Multiplikation der Einzelscores zum Berechnen des Gesamtscores führen würde, sondern eine logarithmierte Form, was wieder Addition der zeichenweisen Scores möglich macht: Sei mit f_i die Wahrscheinlichkeit davon bezeichnet, dass Aminosäure σ_i auftritt. Dann setzt sich zur entsprechenden Matrix M^t die Matrix PAMt zusammen aus:

$$(\text{PAM}t)_{ij} = \log \frac{M^t_{ij}}{f_i} \tag{3.12}$$

BLOSUM-Substitutionsmatrizen

Ein anderer Ansatz beruht nicht auf vorgegebenen phylogenetischen Bäumen, sondern auf Blöcken vorgegebener Multiple Alignments. Zur Berechnung wird in diesen gemeinsamen Sequenzabschnitten jede Spalte einzeln untersucht und alle Paare von Aminosäuren und deren Häufigkeiten in der Spalte bestimmt. Sei die Häufigkeit des Paares aus σ_i und σ_j mit f_{ij} bezeichnet, dann ist die beobachtete Wahrscheinlichkeit q_{ij} für das Auftreten des Paares (σ_i, σ_j):

$$q_{ij} = \frac{f_{ij}}{\sum_{\mu=1}^{20} \sum_{\nu=1}^{\mu} f_{\mu\nu}} \tag{3.13}$$

Die Wahrscheinlichkeit für das Vorkommen von σ_i in einem beliebigen Paar ist dann:

$$p_i = \sum_j q_{ij} \tag{3.14}$$

Die erwartete Wahrscheinlichkeit e_{ij} für das Auftreten eines Paares aus σ_i und σ_j ist dann $e_{ij} = p_i p_j + p_j p_i = 2p_i p_j$ für $i \neq j$ bzw. $e_{ii} = p_i^2$. Diese Werte werden nun noch logarithmiert, mit einem Skalierungsfaktor versehen, auf ganze Zahlen gerundet und ergeben so die Werte der BLOSUM-Matrix. Mit einer Zusammenfassung ähnlicher Sequenzen wird dann der starke Einfluß vieler ähnlicher Sequenzen auf das Ergebnis reduziert. Die Zahl in der Bezeichnung der BLOSUM-Matrix gibt an, ab welcher Identität die Sequenzen als gleich betrachtet werden und zusammengefasst werden. So wurden bei der Bestimmung der BLOSUM62-Matrix alle Sequenzen zusammengefasst, die im betrachteten Block eine Übereinstimmung von mehr als 62% der Aminosäuren aufweisen.

Die Daten zur Berechnung der BLOSUM-Matrizen stammten aus der Swiss-Prot Proteindatenbank und wurden ausgehend von einem sehr einfachen Scoringsystem durch iteratives Alignment bei gleichzeitiger Berechnung und Verwendung der entsprechenden BLOSUM-Matrix erstellt.

Mit der Vorstellung von BLOSUM wurde auch gleich ein Vergleich zu anderen Scoringsystemen, unter anderem auch den PAM-Matrizen durchgeführt - mit dem Resultat, dass die BLOSUM62-Matrix eine deutliche Verbesserung gegenüber den bis dahin verwendeten Systemen darstellt.

Da im Programm MSA die PAM250-Matrix verwendet wird, wurden im Vergleich in dieser Arbeit zunächst sowohl die PAM250- als auch die BLOSUM62-Matrix verwendet. Dabei war festzustellen, dass sich die beiden Systeme in der Größe des Scores und den Größen der Scoreunterschiede zwischen Verfahren und Datensätzen unterscheiden, aber sonst kein unterschiedliches Ranking liefern. Wegen der besseren Erkennbarkeit von Unterschieden bei den mit BLOSUM62 berechneten Scores wurde diese gewählt und die PAM250-Matrix nicht für die Auswertung verwendet. Diese Abstufung der Aussagekraft der beiden Scoring-Systeme wurde schon bei einem Alignmentvergleich in [Got96] festgestellt, wo auch eine Betrachtung der Leistung der Scoringsysteme durchgeführt wurde. Da die PAM-Matrizen durch die Untersuchung von Globinen aufgestellt wurden, sind sie für diese Klasse von Proteinen gut geeignet. Für andere Proteine werden deshalb BLOSUM62 oder weitere, spezialisierte Scoring-Systeme eingesetzt.

Zur Bewertung der Alignments wird wieder die Summierung zum SPScore gewählt, um von dieser paarweisen Sequenzbewertung zur Bewertung des Multiple Alignment zu kommen. Der Parameter mit dem größten Einfluss ist hier die Gap-Penalty. Um ein einfaches Modell zu erhalten, wird hier eine lineare Gap-Penalty gewählt, also mit demselben Wert für Gap-Open-Penalty und Gap-Extension-Penalty. Bei der Wahl dieses Wertes ist leider nicht offensichtlich, welche Größe sinnvoll ist und welche zu falschen Bewertungen der Alignments führen. Da alle Verfahren ein unterschiedliches Modell der Gap-Penalty haben, gibt es von dieser Seite auch wenig Hinweise. Für den praktischen Test wur-

de als Gap-Penalty für das Scoring-System der Durchschnitt aller Mismatch-Bewertungen verwendet. Dies ist beispielsweise bei BLOSUM62 $g = -1$ und liefert auch brauchbare und gut interpretierbare Ergebnisse.

3.3.3 Gewichtung mit Sekundärstrukturinformationen

Die Abfolge der Aminosäuren in Proteinen wird als Primärstruktur bezeichnet und gerade diese wird in Multiple Alignments betrachtet. Wichtiger für die Funktion ist dagegen die Sekundärstruktur, von denen die bekanntesten Vertreter die α-Helix und das β-Faltblatt sind. Diese sind für Teile der dreidimensionalen Gestalt, der Faltung, und damit der Funktion des Proteins verantwortlich. Beide bilden sich durch die Seitenketten der Aminosäuren aus. Die α-Helix ist ein starres, spiralförmiges Gebilde und dient der festen Struktur in Proteinbereichen. Das β-Faltblatt bildet dagegen Bindungen zu anderen β-Faltblättern im Protein aus.

Die Bestimmung der Strukturen kann auf mehrere Arten erfolgen: Die traditionelle Röntgenstrukturanalyse ist sehr aufwändig und besteht aus einer direkten physikalischen Untersuchung des Proteins. Eine andere Möglichkeit bieten Sekundärstruktur-Vorhersage-Verfahren, die anhand bestimmter Kombinationen von Aminosäuren eine Vorhersage der Struktur durchführen. Diese sind aber noch vergleichsweise unsicher.

Weitere 3D-Strukturen in Proteinen sind die Tertiärstruktur, die die Anordnung der Sekundärstrukturen beschreibt und die Quartärstruktur, die den Gesamtaufbau des Proteins darstellt. Diese anhand der Sequenzen zu erkennen ist Gegenstand aktueller Forschung.

Beim Verfahrensvergleich wurden die in BAliBASE teilweise vorhandenen Informationen über die Sekundärstruktur eingesetzt, um die Bewertung zu verfeinern. So ist eine Mutation, die eine Sekundärstruktur und damit auch die Funktion des Proteins verändert, eher unwahrscheinlich. Da sich aber nicht alle Mutationen auf die Sekundärstruktur auswirken, bräuchte man in diesen Bereichen andere Scoring-Systeme. Da diese nicht vorliegen, wurde eine andere Alternative gewählt: In Bereichen der Sekundärstruktur werden einfach alle Scores (d.h. sowohl die Distanz als auch die durch Substitutionsmatrizen berechneten) stärker gewichtet, um dort Mutationen stärker zu bestrafen und richtiges Alignment besser zu bewerten. Da sich an allen Positionen innerhalb der Sekundärstruktur Änderungen negativ auswirken können, wurde eine sehr einfache Gewichtungsfunktion gewählt: Für jeden Vergleich von zwei Aminosäuren σ_1 und σ_2 wird deren Sekundärstruktur betrachtet und der Score dieser beiden Zeichen für jedes Zeichen σ_i mit dem Faktor s multipliziert, dass sich innerhalb einer bekannten Sekundärstrukturregion befindet. Befinden sich also beide Zeichen innerhalb von Sekundärstrukturen, lautet der Gewichtungsfaktor: s^2. Im Vergleich wurde $s = 2$ gewählt, da dieser Faktor schon deutliche Unterschiede zur ungewichteten Version liefert.

3.3.4 Gruppenweise Bewertung

Da manche der Referenzgruppen von BAliBASE verschiedene Familien von Proteinen oder weit entfernte Sequenzen enthalten, soll nicht nur untersucht werden, wie gut das Alignment von allen Sequenzen ist und wie sich die Hinzunahme von Sequenzen aus anderen Familien auf die Qualität des Gesamtalignments auswirkt, sondern auch wie gut das Alignment innerhalb der Gruppen bzw. Proteinfamilien ist. Um zu klären, ob Verfahren diese Gruppen erkennen und innerhalb dieser dann ein gutes Alignment erreichen, bietet es sich an, die Bewertung auf die Gruppen einzuschränken. In dieser Arbeit wurden bei einer Bewertung mit Berücksichtigung der Proteinfamilien, zeichenweise Vergleiche über Familiengrenzen hinweg ignoriert werden und die zeichenweisen Vergleiche innerhalb der Familien wie üblich gezählt und addiert werden, was dann zu einem modifizierten SPScore führt.

3.3.5 Programm MScore

Alle oben beschriebenen Bewertungsfunktionen wurden in einem Programm mit dem Namen „MScore“ implementiert. Es wurde ein objektorientierter Ansatz gewählt, da damit eine einfache Verbindung der verschiedenen Gewichtungen mit dem Differenzscore oder unterschiedlichen Scoring-Systemen möglich ist. Als wichtigste Scoring-Systeme werden unter anderem eine Reihe von PAM-Matrizen und BLOSUM-Matrizen unterstützt, welche dem CLUSTAL-W-Paket entnommen wurden. Neben den einfachen Gewichtungen aus den Abschnitten 3.3.3 und 3.3.4 werden weitere Gewichtungen unterstützt, die in das Programm eingebaut wurden, aufgrund mangelnder Aussagekraft jedoch nicht in den Test einbezogen wurden.

Sekundärstrukturen können je nach Art unterschiedlich gewichtet werden und es kann eine einfache stückweise lineare anstelle von einer konstanten Gewichtung verwendet werden, die dann etwa den Anfang und das Ende der Struktur gar nicht oder geringer gewichtet.

Proteinfamilien lassen sich auch mit Gewichtungsfaktoren versehen, beispielsweise könnten Scores zwischen Gruppen mit 0.5 und in Gruppen mit 2 gewichtet werden. Der im Test verwendete Fall entspricht dann den Gewichtungsfaktoren 0 und 1. Ein erfolgloser Ansatz war die Erkennung von Gruppen durch die Scores der paarweisen Alignments und dann die Gewichtung mit Hilfe dieser Scores. Dabei gab es nämlich Gruppen mit niedrigeren paarweisen Scores innerhalb der Gruppen als zwischen den Gruppen.

MScore bietet ein kommandozeilen-orientiertes Benutzer-Interface, das die Einbindung in Skripte zur automatischen Ausführung gestattet. Dabei sind neben dem zu bewertenden Alignment im GCG/MSF-Format auch falls gewünscht eine FTB-Sekundärstruktur-Informations-Datei und ein Referenzalignment anzugeben. Nähere Informationen zum Programm und dessen Aufruf sind im Anhang gegeben.

3.4 Beschreibung der Testdurchführung

Der Grobablauf des Tests läßt sich kurz darstellen: Zunächst wurde BAliBASE anhand der Übersicht im HTML-Format bereinigt. Es wurden alle in dieser nicht erwähnten Alignments sowie doppelt vorhandene Verzeichnisse entfernt. Dann wird für jedes zu untersuchende Verfahren

1. das Referenzalignment in Sequenzen umgewandelt, indem sämtliche Gaps entfernt werden,
2. dann die Sequenzen in das Eingabeformat des zu untersuchenden Programms gebracht,
3. das Programm mit dem Datensatz gestartet und
4. das Ergebnis wieder in ein GCG/MSF-Alignment umgewandelt

Dabei wird in Schritt 3 das Programm mit dem „time“-Kommando von UNIX gestartet, was eine Bestimmung der Laufzeit ohne Beeinflussung durch andere Prozesse erlaubt. Alle Verfahren wurden auf diese Art auf einem Linux-Rechner mit AMD 1,4GHz-Prozessor und 512MByte Hauptspeicher getestet, was eine direkte Vergleichbarkeit gewährleistet. Die Hauptspeichergröße von 512MByte war die Grenze für die Programme, die mangels Speicherplatz abgebrochen wurden.

Im nächsten Schritt wurden zu allen Alignments die Scores mit dem Programm MScore berechnet und zwar in jedem Verzeichnis die folgenden:

1. Distanz vom Referenzalignment,
2. Distanz vom Referenzalignment mit Sekundärstrukturgewichtung,
3. Distanz vom Referenzalignment mit Sekundärstrukturgewichtung und gruppenweiser Bewertung,
4. BLOSUM62,
5. BLOSUM62 mit Sekundärstrukturgewichtung und
6. BLOSUM62 mit Sekundärstrukturgewichtung und gruppenweiser Bewertung

Die Scores mit Gewichtung wurden nur in den Referenz-Gruppen berechnet, bei denen entsprechende Informationen vorliegen, nicht in den Gruppen 6 bis 8. Zusätzlich wurden alle BLOSUM62-Scores zu den Referenzalignments bestimmt.

Aus diesen Daten wurden mit Gnuplot Graphen erzeugt, die die Ergebnisse eines Verzeichnisses, also einer Referenzgruppe, und die Alignment-Parameter sowie die Laufzeiten der Verfahren enthalten. Die interessanten Fälle sind im nächsten Abschnitt zu sehen.

3.5 Auswertung nach BAliBASE-Datengruppen

Im folgenden Abschnitt werden die mit den in Abschnitt 3.3 vorgestellten Bewertungsfunktionen berechneten Scores und Laufzeitstatistiken nach den BAliBASE-Referenzgruppen ausgewertet. Dabei sind zu jeder Gruppe die Alignmentparameter, die Anzahl der Sequenzen und die Längen der längsten und kürzesten Sequenz in einem Graphen angegeben, da sich diese direkt auf die Scores und Laufzeiten auswirken. Neben den Laufzeiten der verschiedenen Programme sind auch die Scores für die Testdatensätze angegeben. Sind die Darstellungen von ungewichteten und gewichteten Scores nahezu identisch, so wurde die Darstellung einer Version weggelassen. Dies ist auch etwa bei den Referenzgruppen 5 bis 8 der Fall, bei denen keine Gruppen- und Sekundärstrukturinformationen im verwendeten Format vorliegen.

Bei der Darstellung der Graphen sind die Testdatensätze immer mit ihren Bezeichnungen auf der x-Achse aufgetragen. Bei **alp** sind in den drei Kurven die Anzahl der Sequenzen der einzelnen Datensätze (Skala rechte y-Achse) sowie die Länge der jeweils längsten und kürzesten Sequenz („lang“ bzw. „kurz“, Skala linke y-Achse) aufgetragen. In den Distanz- bzw. Scoregraphen sind die Bewertungen zu den verschiedenen Verfahren angegeben, wobei bei den BLOSUM62-Scores noch der Score des Referenzalignments dargestellt ist (Bezeichnung „BAliBASE“).

Bei den Scorebetrachtung muss berücksichtigt werden, dass sowohl Differenz-Scores als auch BLOSUM62-Scores mit dem Produkt aus Sequenzanzahl und -längen anwachsen. In der folgenden Beschreibungen werden solche allgemeinen Beobachtungen nur bei der ersten Gruppe erwähnt, bei der sie auftreten.

Abkürzung	Bedeutung
dis	Distance Score
	Differenz-Score (Abschnitt 3.3.1)
sdis	Secstr. Distance Score
	dis mit Sekundärstruktur-Gewichtung (Abschnitt 3.3.3)
csdis	Cut Secstr. Dist. Score
	sdis mit gruppenweiser Bewertung (Abschnitt 3.3.4)
blosum	BLOSUM62 (Abschnitt 3.3.2)
sblosum	Secstr. BLOSUM62
csblosum	Cut Secstr. BLOSUM62
alp	Alignment Parameter
ts	Time Stats - Laufzeitstatistiken

Tabelle 3.1: Abkürzungen der verschiedenen betrachteten Scores

In den nächsten Abschnitten gibt es zu jeder besprochenen Gruppe eine Abbildung mit den Graphen der Ergebnisse, die aus Platzgründen unter Umständen auf der nachfolgenden Seite zu finden ist. Dabei werden für die Scores und den ihnen zugeordneten Graphen die in Tabelle 3.1 angegebenen Abkürzungen

verwendet, welche auch mit den Dateinamen der zugehörigen Dateien auf der beigefügten CD-ROM übereinstimmen.

3.5.1 Referenz 1

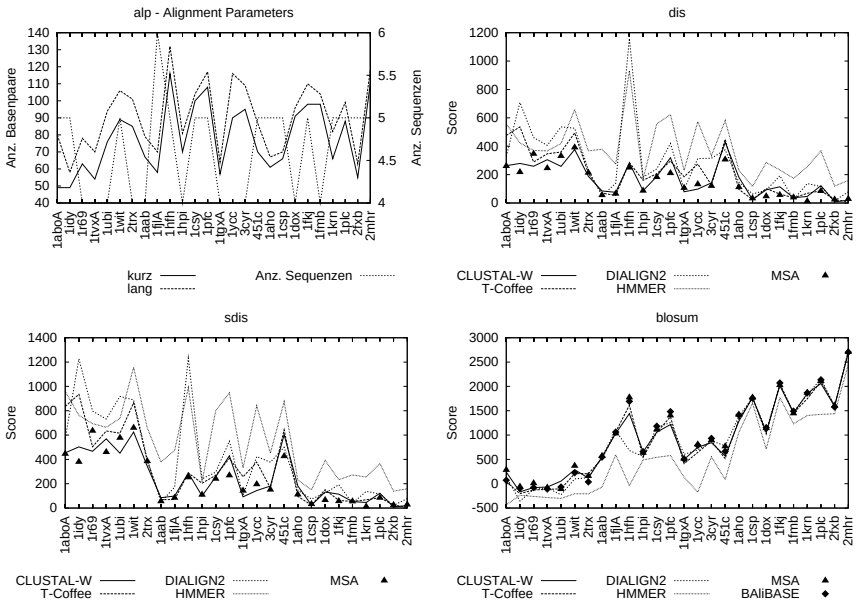


Abbildung 3.7: Ergebnisse Referenz 1, kurze Sequenzen

In allen drei Gruppen in dieser Referenz ist auffällig, dass bei den unähnlichen, also in den Graphen im linken Drittel dargestellten Alignments, der **sdis**-Score etwa doppelt so groß wie der **dis**-Score ist, was nicht weiter verwunderlich ist, da in diesen Fällen die Information in den Sequenzen nicht ausreicht, um ein gutes Alignment zu erhalten. Ein ähnlicher Effekt ist bei **blosum** und **sblosum** zu beobachten: Bei den langen und ähnlichen Sequenzen (rechtes Drittel der Graphen in Abbildung 3.9) werden Bereiche mit Sekundärstruktur von allen Programmen gut aligned und der Score dort ist etwa 40% höher, während er in den anderen Fällen gleich ist. Dies tritt erst bei den längeren Sequenzen auf, da dort auch mehr Aminosäuren mit Sekundärstruktur-Information vorhanden sind.

Nun zu den Ergebnissen einzelner Verfahren:

Bei MSA versagt der Relevanztest des Verfahrens bei Sequenzen mittlerer Länge, die weit entfernt verwandt sind: Hier funktioniert die Abschätzung der unteren Schranke offenbar nicht, so dass eine zu kleine Schranke gewählt wird. In den anderen Fällen ist MSA - wie erwartet - das Verfahren, dass am dichtesten am Referenzalignment liegt (**dis** und **sdis**) und außerdem immer den optimalen

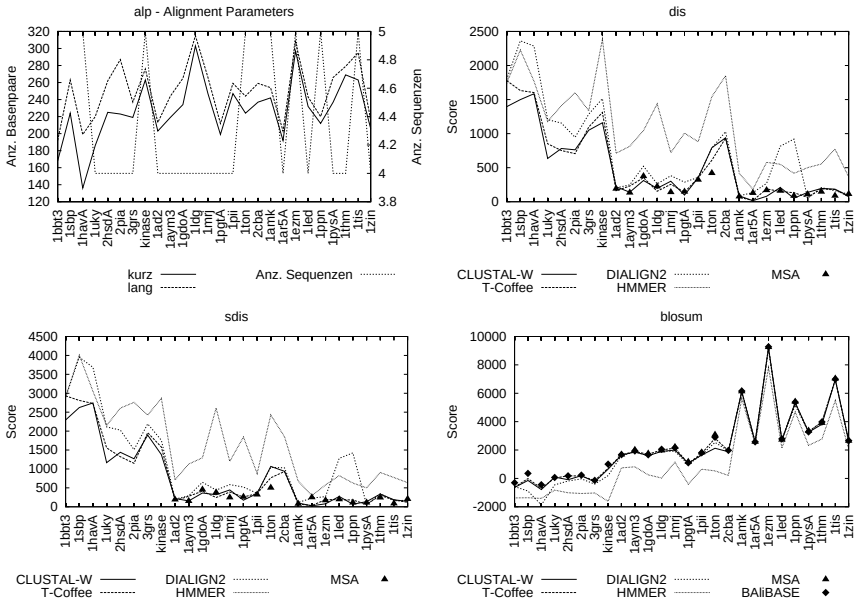


Abbildung 3.8: Ergebnisse Referenz 1, mittellange Sequenzen

Score des Referenzalignment liefert.

DIALIGN liefert Alignments, die etwas weiter entfernt vom Referenzalignment sind als die von CLUSTAL-W oder T-Coffee, aber bis auf den Datensatz „1havA“ der mittel langen Sequenzen nicht sehr viel schlechter sind. Die niedrigen **blosum**- und **sblosum**-Scores bei diesem Alignment lassen sich dadurch erklären, dass DIALIGN lokale, stark konservierte Bereiche aligned und dabei viele Gaps entstehen, die im Referenzalignment an anderen Stellen entstehen. Ausserdem entstehen dabei sehr viele kurze Blöcke, die jeweils durch wenige Gaps getrennt sind. Dies ist das typische Problem von DIALIGN und tritt immer wieder auf.

Für diese Klasse von Alignments sind insgesamt MSA am besten und danach CLUSTAL-W und T-Coffee geeignet.

3.5.2 Referenz 2 - zusätzliche entfernt verwandte Sequenzen

In dieser Referenz-Gruppe führt die Einbeziehung der Sekundärstrukturinformationen zu keinen neuen Erkenntnissen, dafür liefert aber die Information über die Familienzugehörigkeit der Sequenzen deutliche Ergebnisse.

Wie in **sblosum** und **csblosum** in Abbildung 3.10 deutlich zu sehen ist, fallen bei einer Betrachtung der Familien die Scores nicht mehr so unterschiedlich aus wie bei der Betrachtung des Gesamtalignments. Die meisten Verfahren wer-

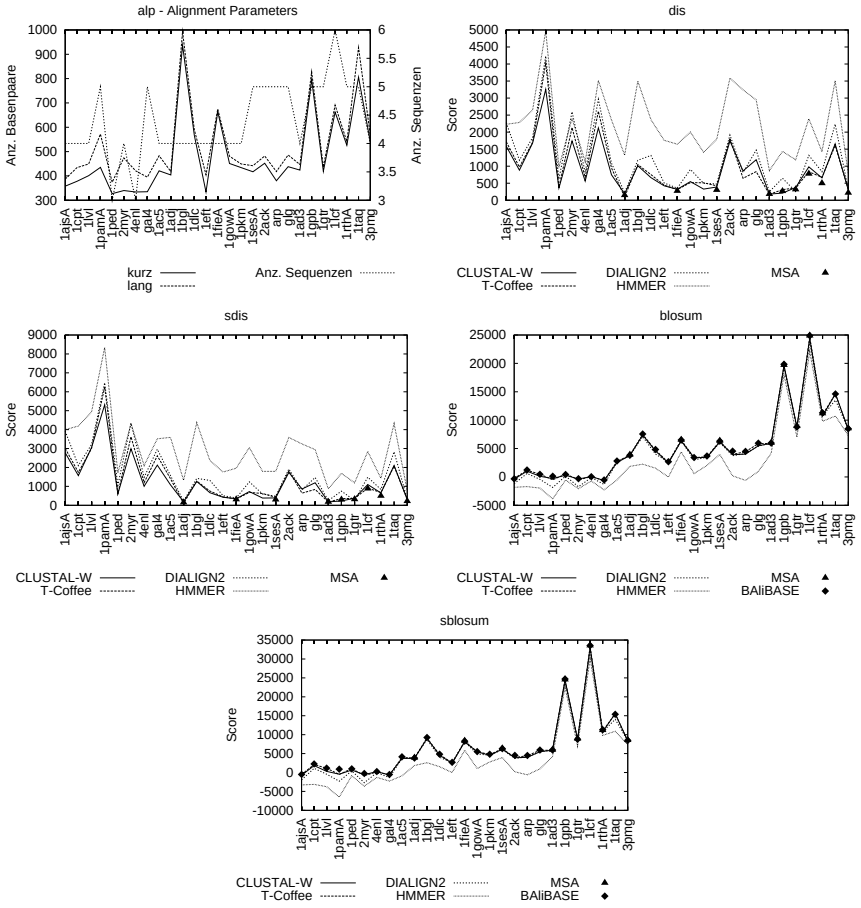


Abbildung 3.9: Ergebnisse Referenz 1, lange Sequenzen

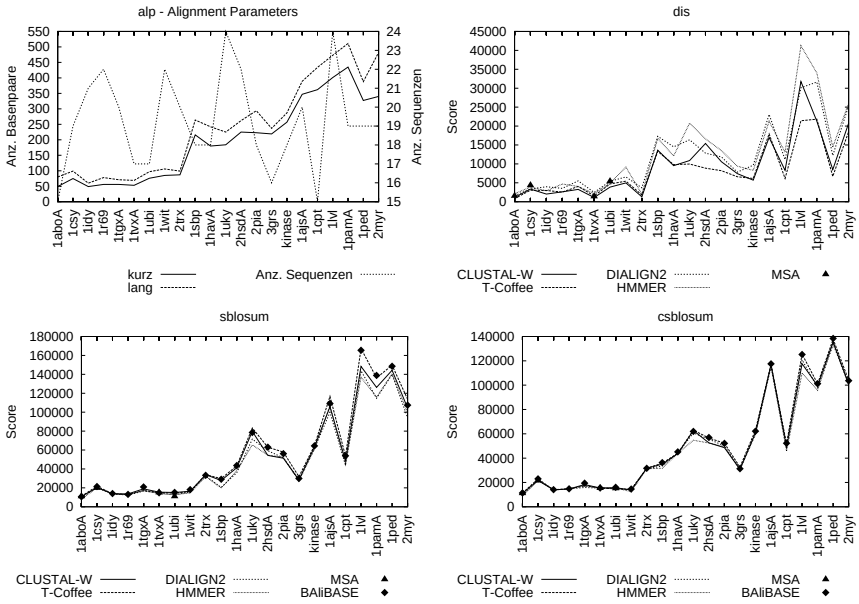


Abbildung 3.10: Ergebnisse Referenz 2

den also durch die Anwesenheit der weit entfernt verwandten Waisensequenzen negativ beeinflußt, wobei T-Coffee noch am besten abschneidet: Dieses Verfahren liefert immer einen gleich hohen Score wie das Referenzalignment.

Dass T-Coffee und CLUSTAL-W von der Hinzunahme der Waisensequenz relativ unbeeinflusst bleiben, wird bei einer Betrachtung der Verfahren klar: Diese Sequenzen werden erst vom Alignmentprozeß ausgeschlossen und ganz am Ende zum Alignment hinzugenommen.

HMMER erreicht hier auch mit wenigen Trainingsdaten ein gutes Ergebnis, da die Sequenzen bis auf die Waisensequenzen alle große Ähnlichkeit aufweisen.

Bei MSA versagt hier in den meisten Fällen der Relevanztest und in den anderen Fällen wird ein Alignment bestimmt, dass zwar einen optimalen Score hat, aber deutlich weiter als in Referenz 1 vom Referenzalignment entfernt ist.

Für diese Art von Alignments sind also deutlich T-Coffee und mit Einschränkungen CLUSTAL-W geeignet.

3.5.3 Referenz 3 - mehrere entfernt verwandte Familien

Im Fall mehrerer entfernt verwandter Familien von Proteine wird bei den meisten Verfahren wieder ein deutlich besseres Alignment in den Familien als zwischen diesen erkennbar (Abbildung 3.11, **dis** und **csdis**). Dieser Effekt tritt nur bei CLUSTAL-W ist er verstärkt auf.

Der **sblosum**-Score von T-Coffee ist sogar noch größer als der des Referenz-

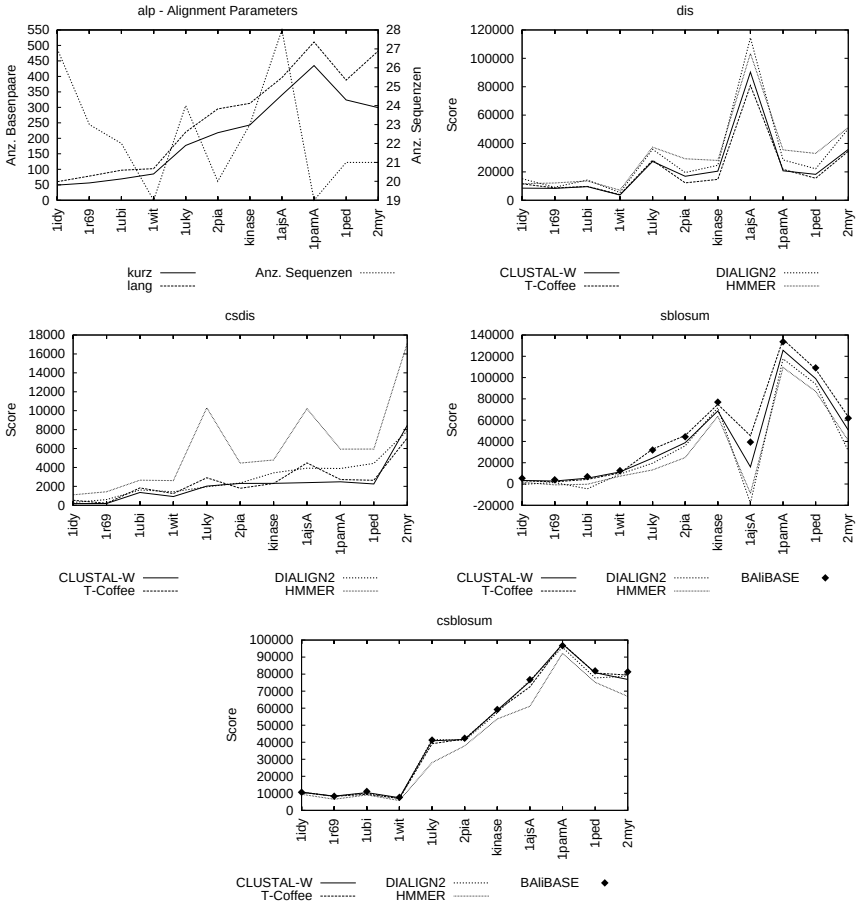


Abbildung 3.11: Ergebnisse Referenz 3

alignments, da dieses die konservierten Bereiche mit bekannter Sekundärstruktur besser aligned.

HMMER zeigt Probleme beim Alignment zwischen den Gruppen und dadurch auch ein schlechtes Alignment in den Gruppen (**sblosum** und **csblosum**), was darauf zurückzuführen ist, dass ein HMM verschiedene Gruppeneigenschaften gleichzeitig modellieren soll.

Bei MSA schlägt in den meisten Fällen der Relevanztest fehl, während in den anderen der Speicherplatzbedarf zu hoch ist, um die Berechnung durchzuführen.

3.5.4 Referenz 4 - endständige Erweiterungen

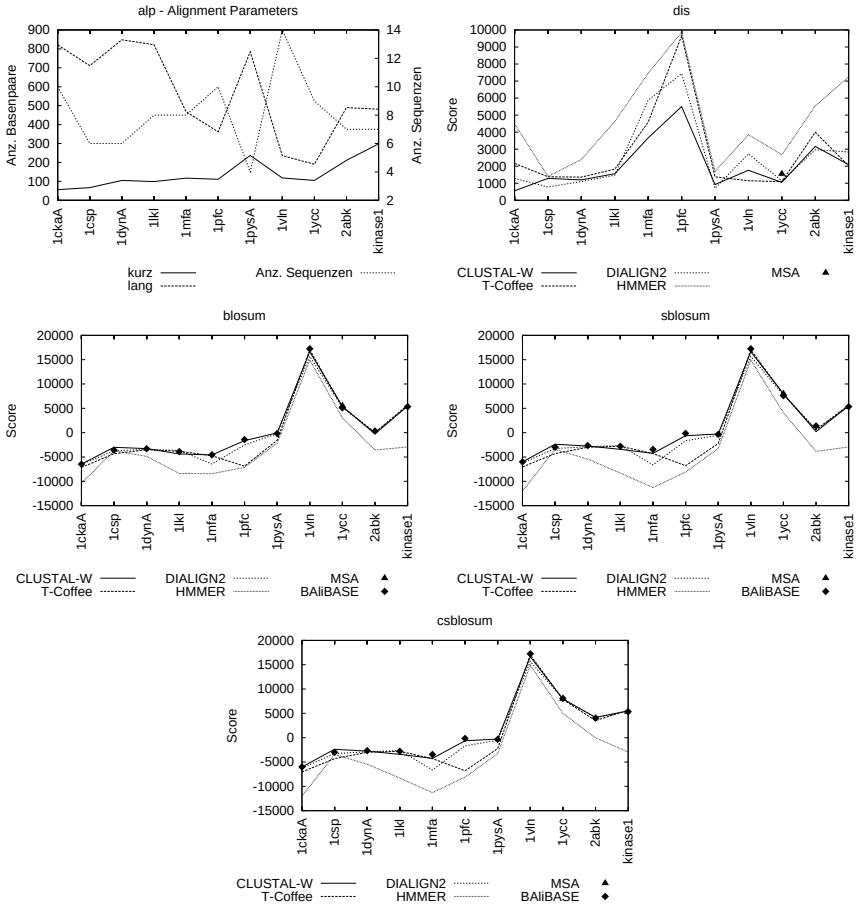


Abbildung 3.12: Ergebnisse Referenz 4

In Referenz 4 ist kaum unterschiedliches Verhalten der Verfahren feststellbar mit den Ausnahmen:

Für MSA sind die meisten Alignments zu groß, also ist keine Berechnung möglich.

HMMER wird offenbar durch die stark unterschiedliche Länge der erweiterten Sequenz beeinflußt und berechnet deshalb meistens kein gutes Alignment, welches sowohl im Gesamtalignment als auch ohne Berücksichtigung der erweiterten, entfernt verwandten Sequenz der Fall ist.

T-Coffee liefert teilweise auch schlechte Alignments, was auf nicht zum optimalen Alignment passende paarweise, lokale Alignments in der Bibliothek hindeuten könnte.

CLUSTAL-W liefert durchgehend gute Alignments, da die erweiterte Sequenz erst am Ende zum Alignment hinzugenommen wird.

3.5.5 Referenz 5 - Insertionen

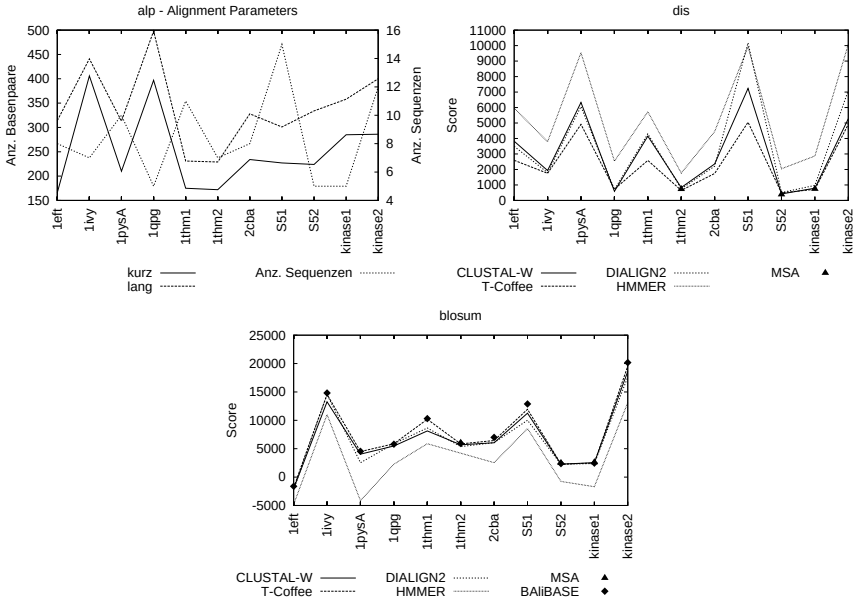


Abbildung 3.13: Ergebnisse Referenz 5

Obwohl es sich bei Insertionen in Sequenzen um eine gänzlich andere Problemstellung als bei der Erweiterung handelt, treten hier ähnliche Ergebnisse wie in Referenz 4 auf. Nur bei DIALIGN und vor allem T-Coffee ist eine Verbesserung festzustellen, die auf die größere Bedeutung von lokalen Alignments hindeutet, wenn es darum geht, mehrere Insertionen zu berücksichtigen. Möglicherweise würde eine lokale Scoringfunktion, die lokale Ähnlichkeiten bewertet,

bessere Resultate liefern.

Diese Resultate widersprechen nicht den in [TPP99b] vorgestellten, wobei dort noch zusätzliche Scores verwendet wurden, die mehr auf Besonderheiten der Referenzen 4 und 5 eingehen.

3.5.6 Referenz 6 - Wiederholungen

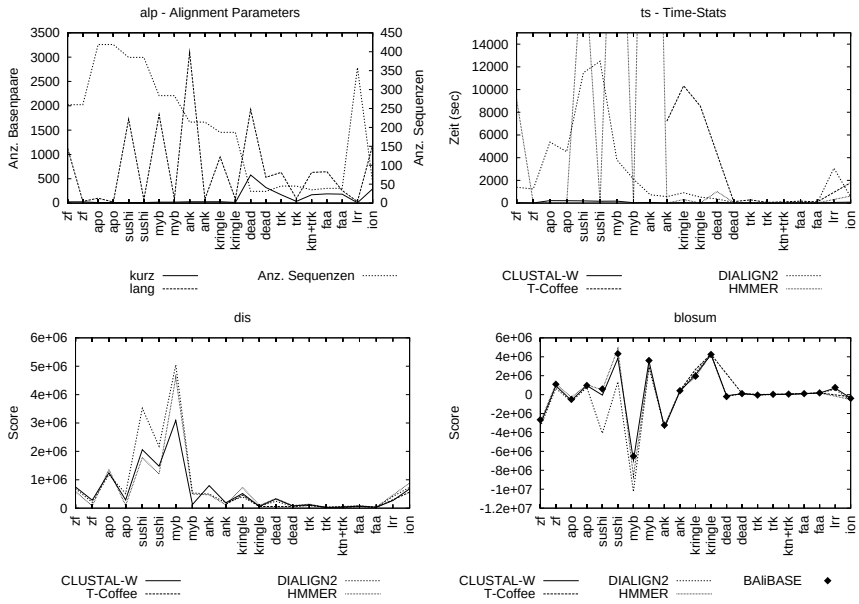


Abbildung 3.14: Ergebnisse Referenz 6

Diese Referenz-Gruppe besteht aus Alignments mit Wiederholungen unterschiedlicher Art, die leider nicht genau dokumentiert sind. Dabei sind die Alignments der Sequenzen unter „Referenz 6“ und die Alignments der wiederholten Sequenzbereiche in den Untergruppen von Referenz 6 zu finden. Bei manchen dieser Gruppen treten mit den verwendeten Scoring-Funktion keine deutlichen Ergebnisse hervor, so dass diese weggelassen wurden. Hier wäre eine Scoring-Funktion nötig, die keinen lokalen, zeichenweisen Ansatz verfolgt und somit auch Wiederholungen entsprechend bewerten kann.

Da die Alignments in dieser Gruppe umfangreicher sind, also mehr und längere Sequenzen enthalten, ist der Speicherplatzbedarf von T-Coffee zu hoch und führt in manchen Fällen dazu, dass kein Alignment berechnet werden kann. Für andere Alignments muss mit sehr hohem Zeitaufwand gerechnet werden.

DIALIGN hat mit den verschiedenen Wiederholungen in dieser Referenz-Gruppe viele Probleme und falsche Zuordnungen in lokalen Alignments führen dabei zu schlechten Alignments.

Dem HMMER-Verfahren kommt hier die große Datenmenge zu gute, was ein genaues Trainieren des Modells ermöglicht und damit zu guten Alignments führt.

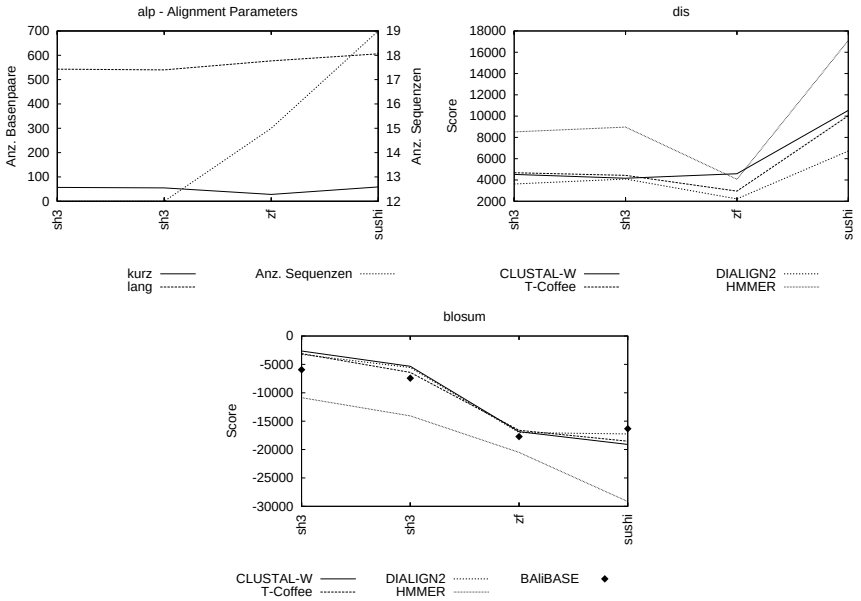


Abbildung 3.15: Ergebnisse Referenz 6, Gruppe 1b

Gruppe 1b

Das Auffälligste in der Gruppe 1b ist das Alignment „zf“, das sowohl einen niedrigen Score, als auch eine niedrige Distanz zum Referenzalignment hat. Dieser Sachverhalt ensteht aus den sehr kurzen wiederholten Bereichen im Alignment, zu denen noch viele eher zufällige Zeichen kommen. Dadurch finden alle Programme die kurzen wiederholten Bereiche, können aber kein gutes Alignment der anderen bestimmen. In dieser Gruppe der Alignments mit mehrfach wiederholten Bereiche kommt es ausserdem zu einer uneindeutigen Zuordnung der Wiederholungen, wie **blosum** zeigt: Es gibt bessere Alignments als das Referenzalignment, die trotzdem noch von diesem weit entfernt sind. Interessant ist auch die gute Bewertung von Alignments, die mit DIALIGN erstellt wurden. Dieser Effekt tritt deshalb auf, weil es sich um Alignments handelt, in denen nur die wiederholten Bereiche enthalten sind, also eigentlich lokale Alignments der Sequenzen aus Referenz 6.

Gruppe 2a

Hier kommt es beim Datensatz „myb“ zu einem Versagen von DIALIGN und HMMER: Durch das falsche Alignen der Wiederholungen berechnen diese Programme ein schlechtes Alignment auch für die übrigen Bereiche.

Gruppe 2b

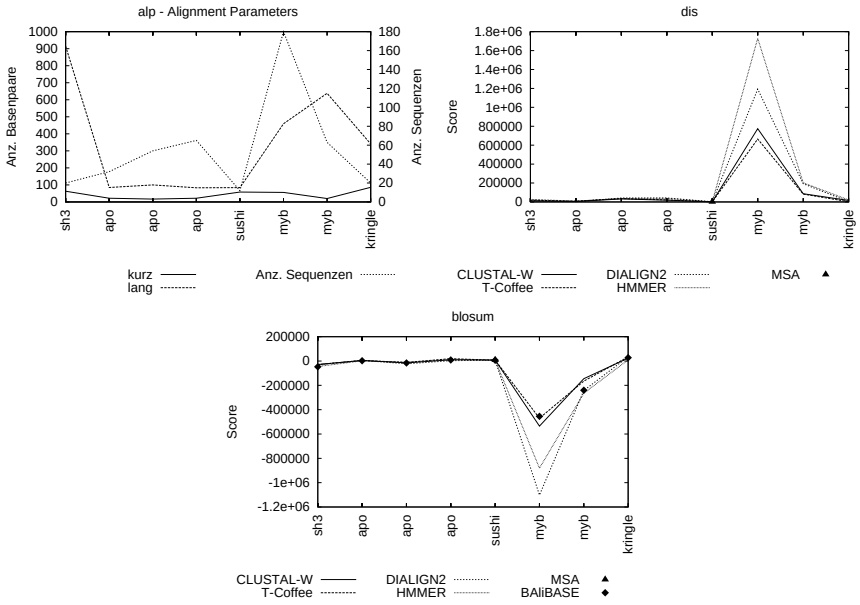


Abbildung 3.16: Ergebnisse Referenz 6, Gruppe 2a

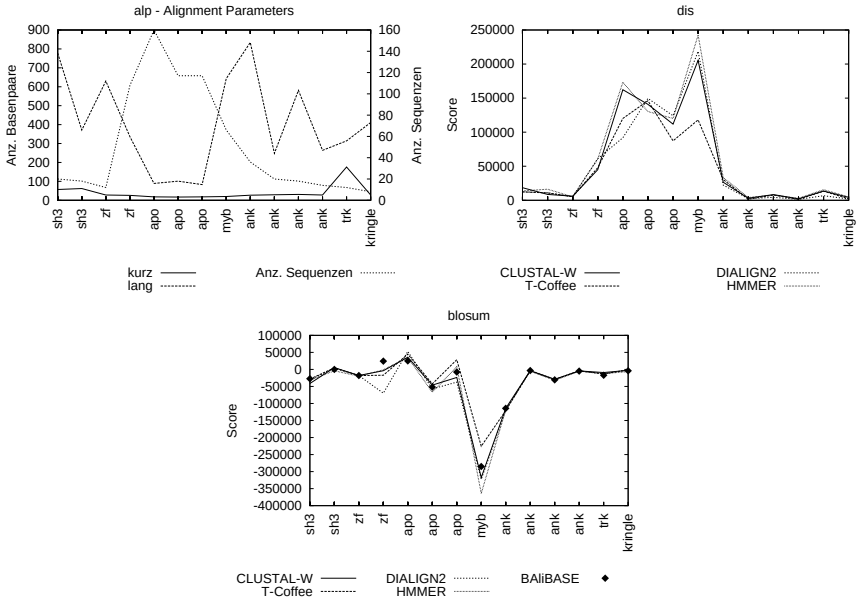


Abbildung 3.17: Ergebnisse Referenz 6, Gruppe 2b

Durch das bessere Alignen der langen Wiederholungen durch T-Coffee kommt es hier zu einem Alignment, das nicht weit vom Referenzalignment entfernt ist und einen besseren Score als dieses hat.

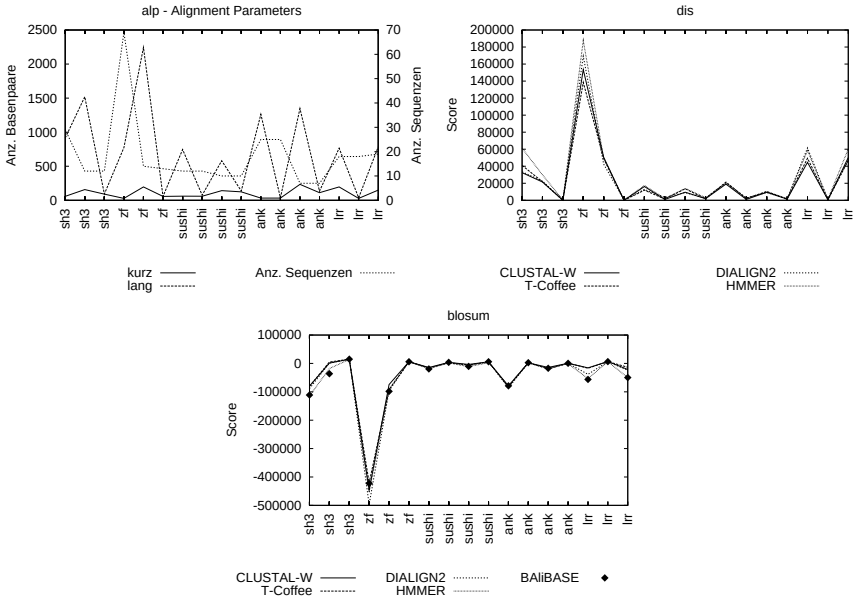


Abbildung 3.18: Ergebnisse Referenz 6, Gruppe 4

Gruppe 4

In Gruppe 4 gibt es keine Besonderheiten bis auf das Alignment „zf“: Wie in **blosum** zu erkennen ist, gibt es hier nicht nur viele Zeichen, sondern auch im Referenzalignment viele Gaps, was natürlich zur Folge hat, dass die Alignments der Programme auch weit von diesem entfernt sind.

3.5.7 Referenz 7 - transmembrane Proteine

Bei den transmembranen Proteinen haben lokale Alignmentverfahren wie DIALIGN Probleme, da die transmembranen Strukturen relativ kurze, konservierte Bereiche sind. Diese werden dann aufgrund besserer und dem Referenzalignment widersprechender lokaler Alignments in anderen Bereichen auseinandergeschoben und treffen so im Alignment nicht mehr aufeinander.

HMMER wird auch von den nichtcodierenden Bereichen beeinflußt, liefert aber dennoch Alignments mit ähnlichen Scores wie die anderen Verfahren.

CLUSTAL-W und T-Coffee sind hier am besten geeignet, obwohl auch hier große Distanzen zum Referenzalignment auftreten. Eine besser an die Situation angepasste Scoring-Funktion mit Bezugnahme auf die Proteinbereiche dürfte wieder bessere Resultate liefern.

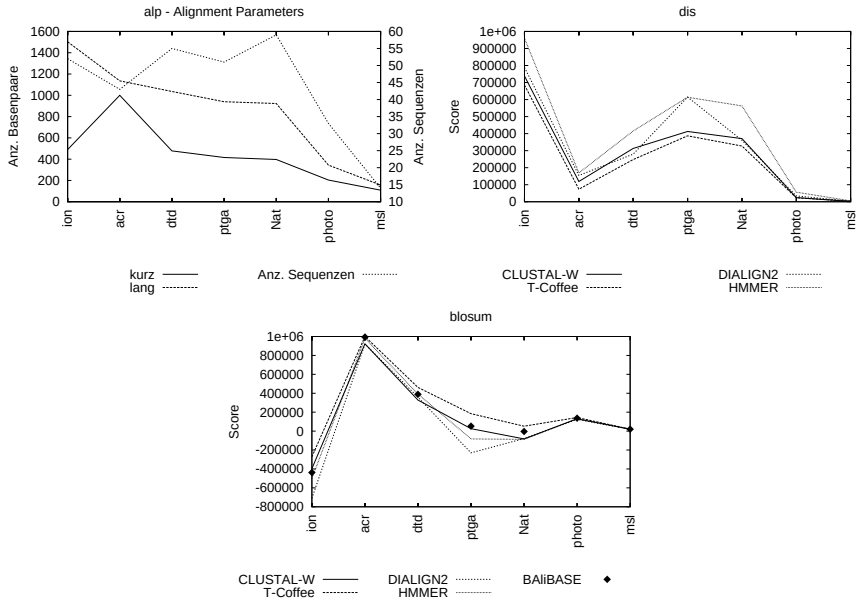


Abbildung 3.19: Ergebnisse Referenz 7

3.5.8 Referenz 8 - Inversionen

Bei diesem Test sind zwei Dinge auffällig: Die ersten beiden Alignments umfassen viele lange Sequenzen. Dementsprechend hoch sind ihre Scores und Differenzscores. Dabei erreichen fast alle Verfahren einen deutlich höheren **blosum**-Score als das Referenzalignment, was jedoch auf einen Fehler in den Daten zurückzuführen ist: Teile der Sequenz „FYN_XIPHE“ treten um ein Zeichen verschoben auf, was zu vielen Mismatches im Referenzalignment führt.

Die zweite Beobachtung ist das schlechte Abschneiden von HMMER trotz hoher Sequenzanzahl und -länge. Der Grund dafür ist darin zu suchen, dass die invertierten und zufälligen Bereiche zu einem falschen HMM führen.

DIALIGN hat auch Probleme mit den bei HMMER problematischen Bereichen.

Eine entsprechende Scoringfunktion würde hier vielleicht zu besseren Resultaten führen, doch bleibt fraglich, was mit Inversionen und Permutationen in der Reihenfolge von Bereichen innerhalb eines Multiple Sequence Alignment geschehen soll.

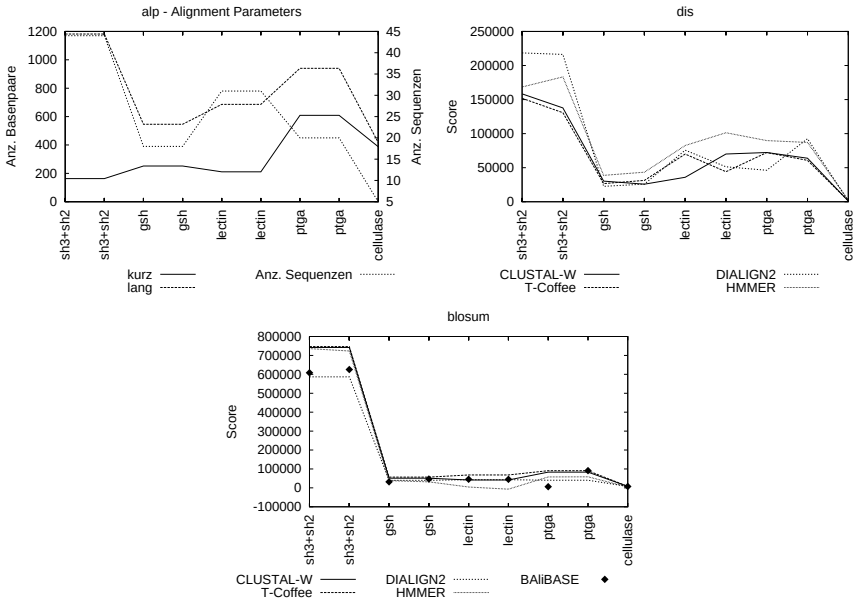


Abbildung 3.20: Ergebnisse Referenz 8

3.6 Zusammenfassung der Vergleichsergebnisse

Bevor genauer auf die Abweichungen von dieser Abschätzung bei den anderen Verfahren eingegangen wird, folgt eine Aufstellung der Gesamtlaufzeit für die betrachteten Verfahren und Datensätze:

Programm	Laufzeit
MSA	24 h, 23 min
CLUSTAL-W	50 min
T-Coffee	33 h, 7 min
DIALIGN	16 h, 54 min
HMMER	55 h, 59 min

Die Laufzeit von CLUSTAL-W wächst ungefähr mit dem Produkt der Sequenzanzahl und der Längen der längsten und kürzesten Sequenz an.

Die kürzere Laufzeit von MSA gegenüber anderen Verfahren erklärt sich dadurch, dass bei großen Datensätzen entweder der Speicherbedarf relativ schnell so groß wurde, dass der Prozess vom Betriebsystem abgebrochen wurde, oder dass bei noch größeren Datensätzen die Anzahl oder Länge der Sequenzen zu hoch war und deshalb das Programm abgebrochen wurde. Hier dürfte der Zeitaufwand in etwa dem in Abschnitt 1.1.2 angegebenen $\mathcal{O}(k^4 2^k n^k)$ entsprechen.

Bei T-Coffee wirkt sich die Erstellung der Bibliothek aus paarweisen Alignments bestimmend auf die Laufzeit und den Speicherbedarf aus. In Referenz 6

befinden sich viele große Datensätze, die zum Teil auch zu einem Programmabbruch wegen Speicherknappheit führen, was am Fehlen des T-Coffee-Graphen in Abbildung 3.14, **ts** bei den ersten Datensätzen zu erkennen ist. Für die anderen Alignments erkennt man dort den teilweise hohen Zeitaufwand.

Bei DIALIGN ist Abbildung 3.14, **ts** gut die Abhängigkeit von den Sequenzanzahlen und -längen zu beobachten, wobei diese Abbildung die Vermutung nahe legt, dass die Laufzeit nicht exponentiell von der Sequenzanzahl abhängt.

Bei HMMER wächst der Zeitaufwand stark mit der Länge der Sequenzen an, was auf die Bestimmung von zu den Sequenzen gehörenden Wegen durch das Modell zurückzuführen sein dürfte. Wie in Abbildung 3.14 deutlich wird, ist dadurch die Abhängigkeit von den Sequenzlängen noch stärker als von der Sequenzanzahl.

Allgemein ist eine Abhängigkeit der Scores und der Laufzeit von allen Sequenzlängen naheliegend, da längere Sequenzen mehr Zeichen beinhalten, für die Matches, Mismatches und Gaps betrachtet werden. Deshalb ist auch immer die Länge der kürzesten Sequenz angegeben.

Bei unähnlichen Sequenzen (Referenz 1, jeweils erste Datensätze pro Gruppe) sind wegen der wenigen konservierten Bereiche viele Alignments möglich, die zu optimalem Score führen, aber doch weit vom Referenzalignment entfernt sind. Dies ist ein Hinweis darauf, dass die Betrachtung solcher Sequenzen selten zu brauchbaren Ergebnissen führt.

Besonders betroffen davon ist DIALIGN als lokales Verfahren, aber auch wenn sich nur einige nicht konservierte, also stark unterschiedliche Bereiche in den Sequenzen befinden. Das führt zumeist zu einem Alignment mit sehr kurzen übereinstimmenden Blöcken und vielen, langen Gaps zwischen den Blöcken, die zum Teil nicht zusammengehörende Bereiche zuordnen.

Geringere Probleme mit solchen Bereichen hat HMMER, das dann viele einzelne Zeichen aligned und dazwischen sehr viele Gaps einfügt.

Schlechte Alignments liefern diese beiden Verfahren auch bei Wiederholungen in Sequenzen, die in Referenz 6 betrachtet werden. So kommt bei DIALIGN oft eine falsche Zuordnung zwischen den verschiedenen konservierten, wiederholten Bereichen vor, die dann ein gutes Gesamtalignment verhindert. Gut sind in diesem Bereich die globalen Verfahren wie CLUSTAL-W und T-Coffee. Über MSA kann wegen der Größe der Daten hier keine Aussage getroffen werden. Um jedoch das richtige Alignment von Wiederholungen genau beurteilen zu können, wird eine andere Scoring-Funktion benötigt, die nicht nur lokale Sequenzähnlichkeiten berücksichtigt.

Bis auf MSA liefern alle Verfahren bei der Hinzunahme von wenigen weit entfernt verwandten Waisensequenzen noch gute Alignments, wobei die Alignments der anderen Sequenzen wenig von den Waisensequenzen beeinflußt werden. Hier liefert HMMER trotz der geringen Anzahl Trainingsdaten ein relativ gutes Alignment, da fast alle Sequenzen große Ähnlichkeit aufweisen.

Dieser Effekt verstärkt sich beim Alignment von mehreren Proteinfamilien, die innerhalb der Gruppe große Ähnlichkeit aufweisen und zwischen den Gruppen geringe. HMMER kann davon kein gutes HMM mehr bestimmen, da

darin gleichzeitig mehrere Gruppeneigenschaften simuliert werden müßten und erzeugt deshalb keine guten Alignments.

Ein ähnliches Problem besteht, wenn eine oder mehrere Sequenzen eine stark unterschiedliche Länge besitzen. Durch Zuordnungen der anderen Sequenzen zu falschen Bereichen der längeren Sequenzen versagen hier vor allem DIALIGN und HMMER. T-Coffee ist bei Insertionen zwar am besten geeignet, hat aber bei Erweiterungen von Sequenzen an den Enden Probleme, ein optimales Alignment zu finden. In beiden Fällen ist CLUSTAL-W als globales Verfahren gut geeignet. Bis auf wenige oben beschriebene Fälle liefert T-Coffee als Weiterentwicklung von CLUSTAL-W ähnlich gute oder sogar bessere Ergebnisse als dieses.

Zum Abschluß des Kapitels sind die Ergebnisse nochmals in einer Tabelle zusammengefasst, in der die besonders gute oder schlechte Eignung von Verfahren bezüglich der behandelten Kriterien dargestellt ist, wobei CLUSTAL-W mit CLW, T-Coffee mit T-C, Dialign mit Dia und HMMER mit HMM abgekürzt ist. Dabei kommt die geringe Eignung von MSA für die meisten Gruppen dadurch zustande, dass MSA wegen der Speicherplatzanforderungen nicht in der Lage ist, so große Alignments zu verarbeiten, wie sie in den Gruppen in BAliBASE vorkommen. Ähnlich ist für HMMER ein Nachteil, dass die meisten Referenzalignments aus vergleichsweise wenigen Sequenzen bestehen. Die Bewertungen in der Tabelle geben also nicht unbedingt eine prinzipielle Eignung für die entsprechenden Fälle wieder, sondern nur die Eignung für die betrachteten Alignments des empirischen Vergleichs.

		MSA	CLW	T-C	Dia	HMM
Ref. 1	unähnliche Sequenzen	o	o	o	-	-
	ähnliche Sequenzen	+	o	o	o	-
Ref. 2	eine zusätzliche Seq.	-	+	+	-	o
Ref. 3	Proteinfamilien	-	+	o	o	-
Ref. 4	Endst. Erweiterungen	-	+	o	o	-
Ref. 5	Insertionen	-	o	+	o	-
Ref. 6	Wiederholungen	-	o	o	-	o
Ref. 7	Transmembrane Proteine	-	+	+	-	o
Ref. 8	Inversionen	-	o	o	-	-
+ gut geeignet, o nur bedingt geeignet, - schlecht geeignet						

3.7 Anmerkungen

BAliBASE wurde in [TPP99a] als eine Datenbank von Alignments zum Vergleich von Programmen vorgestellt. Eine erste Untersuchung mehrerer Programme wurde dann in [TPP99b] durchgeführt, woran auch Teile des in dieser Arbeit durchgeführten Vergleichs orientiert sind - unter anderem auch der Differenzscore. Die Gruppeneinteilung von BAliBASE 2.0 ist [BTTP] entnommen, wobei für den Vergleich die leicht aktualisierte Version 2.01 eingesetzt wurde. FSSP ist in [HS96], HOMESTRAD in [MDBO98] und HSSP in [SdDS97] beschrieben. Der VAST-Server geht auf [MGB95] zurück, die PDBsum Datenbank auf [LHM+97].

Die zum Proteinstruktur-Vergleichsprogramm SAP gehörende Veröffentlichung ist [Tay99].

Das Programm MSA liegt inzwischen in der zweiten Version vor. Während in [LAK89] die erste Version nur kurz vorgestellt wurde, liefert [GKS95] eine detaillierte Beschreibung des Verfahrens, wobei neben einer ausführlichen Darstellung des Grundalgorithmus auch die Version für die affin-lineare Gap-Penalty besprochen wird. Die größte Weiterentwicklung zur ersten Version besteht jedoch in einer Betrachtung der Frage, welche Knoten zusätzlich noch aus dem Speicher gelöscht werden können, da sie keinen Beitrag zu einem optimalen Alignment liefern und einer dahingehenden Speicherplatzbedarf-Reduktion des Programms MSA. Der Relevanztest wird auch in Kapitel 3.4 von [SM97] kurz und anschaulich besprochen und wurde von Carillo und Lipman [CL88] entwickelt.

Das Programm CLUSTAL-W ist eines der weitverbreitesten Multiple-Sequence-Alignment-Programme was wohl auch an der einfachen Verständlichkeit und nachvollziehbaren Einbindung von biologischem Zusatzwissen in das Verfahren liegen dürfte. Als Vorlage für die kurze Beschreibung diente [THG94] von den Autoren des Programms. Details über die erwähnten Heuristiken sind in [SN87] (Neighbor Joining) und [BCL87] (Approximierung des Scores paarweiser Alignments) zu finden.

T-Coffee ist das neueste Programme in diesem Vergleich und wurde in [NHH00] vorgestellt. Dort sind auch Details über die angesprochenen weiteren Untersuchungen aufgeführt.

DIALIGN wurde erstmals 1996 in [MDW96] beschrieben und ab diesem Zeitpunkt kontinuierlich weiterentwickelt. Die vorliegende Betrachtung ist zusätzlich noch an [Mor99] orientiert, das die Version 2.0 von DIALIGN beschreibt. Zum Verfahrensvergleich wurde die inzwischen erhältliche Version 2.1 eingesetzt. In [SKM01] wird eine weitergehende Betrachtung von DIALIGN vorgenommen.

Da HMMs in der Bioinformatik eine große Rolle spielen, ist auch viel Literatur zu diesem Themenkomplex vorhanden und daher kommt es, dass die Vorstellung von HMMER sich weniger an der zugehörigen Veröffentlichung [Edd98] von Sean Eddy orientiert, als an der umfassenden Einführung [KBM+93]. Die Beispiele sind [CB00] entnommen. Details zu den Algorithmen sind in [CB00] oder - in etwas anderem Kontext - in [Rab89] zu finden.

Die PAM-Matrizen gehen auf [DSO79] zurück, während die BLOSUM-Matrizen in [HH92] eingeführt wurden.

Die beiden betrachteten Sekundärstrukturen gehen auf Linus Pauling und [PCB51] zurück. Eine Einführung in Sekundärstrukturen und deren Vorhersage ist im Internet zu finden: [CWM]. Die Unwahrscheinlichkeit von Insertionen oder Deletionen in Regionen mit besonderer Sekundärstruktur wurde auch schon im Kapitel 3 von [KG99] beschrieben.

Kapitel 4

Biologische Beurteilung

In diesem Kapitel werden die Aspekte des Multiple Alignment diskutiert, die mehr im Zusammenhang mit der Anwendung in der biologischen Praxis stehen. Neben Ergebnissen vorangehender Kapitel liegen diesem auch Gespräche mit Mitarbeitern von Arbeitsgruppen aus der Molekulargenetik sowie Zoologie und Biophysik zu Grunde.

4.1 Anwendungen des Multiple Alignment

Für ein besseres Verständnis der Probleme beim Multiple Alignment werden hier kurz die bekannten Anwendungen aufgezählt:

1. Erstellung von Phylogenien

   Phylogenien werden beispielsweise mit dem Neighbor-Joining-Verfahren aus den paarweisen Sequenzabständen konstruiert und dann mit einer Maximum-Likelyhood-Heuristik verfeinert. Dafür werden die Zuordnungen der Zeichen durch das Multiple Sequence Alignment als Grundlage für die Bewertung der phylogenetischen Bäume verwendet.

2. Bestimmung von Proteinfamilien

   Hier liefert das Alignment eine Aussage über die Ähnlichkeit der gesamten Sequenzen oder von Teilen, mit der dann eine Zuordnung von Sequenzen zu Familien durchgeführt wird.

3. Aufklärung der 3D-Struktur und Funktion von Proteinen

   Dazu wird ein Alignment der zu untersuchenden Sequenz mit Sequenzen schon bekannter Proteine durchgeführt, zu denen man einen Zusammenhang bezüglich Struktur und Funktion vermutet. Das Multiple Alignment liefert dann eine Zuordnung von Bereichen, wobei auch Bereiche der zu untersuchenden Sequenz zu Bereichen mit bekannter Struktur und Funktion in anderen Sequenzen zugeordnet werden. Daraus versucht man dann auf die Funktion dieser Bereiche in der neuen Sequenz zu schließen.

4.2 Probleme des Multiple Alignment

Ein großes Problem des Multiple Alignment ist das Modell, das hier zum Einsatz kommt: Durch direkte Ähnlichkeitsbetrachtungen der Zeichenfolgen gelangt man zu Aussagen über Ähnlichkeit oder Unähnlichkeit von Proteinen, DNA-Sequenzen, oder auch nur Teilstücken. Da dieses Modell von der gleichen Reihenfolge der Aminosäuren in den Sequenzen ausgeht, versagt es, wenn die Reihenfolge von ganzen Bereichen bzw. Domains oder in Bereichen verändert wird, wie etwa bei Inversionen. Außerdem gibt es Proteine verschiedener eng verwandter Spezies, die eine gänzlich andere Sequenz aufweisen, obwohl die Funktion dieselbe ist. Hier ist als prominentestes Beispiel der rote Blutfarbstoff Hämoglobin zu nennen, der sich von Spezies zu Spezies sehr stark in seiner Proteinsequenz unterscheidet. In diesen Fällen versagt das Multiple Alignment, wenn es etwa um Phylogeniebestimmungen geht, da Ähnlichkeiten in den Sequenzen verwandter Spezies fehlen.

4.3 Mögliche Verbesserungen

Zwar funktioniert das Multiple Alignment sowohl bei DNA-Sequenzen als auch bei Proteinsequenzen, doch zeichnen sich letztere durch eine reichhaltigere Struktur aus. So kann man schon deutliche Unterschiede in Häufigkeit und chemischen Eigenschaften der Aminosäuren feststellen, was dazu führt, dass DNA-Sequenzen meistens erst in Proteinsequenzen umgewandelt werden, um dann diese in ein Multiple Alignment zu bringen. Da jede Aminosäure durch ein Codon von drei Nukleobasen in der DNA codiert wird, benötigt man eine Aussage über den sogenannten Leserahmen, der festlegt, wo die Übersetzung beginnt. Die Bestimmung des „richtigen“ Leserahmen funktioniert meistens noch relativ gut, da bei den anderen Leserahmen ungewöhnliche Proteinsequenzen entschlüsselt werden, doch Insertionen oder Deletionen einzelner Nukleobasen führen zu Fehlern, die nicht so einfach zu erkennen und zu beseitigen sind.

Ähnlich wie man auf diese Art zu den DNA-Sequenzen noch mehr Informationen erhalten kann, kann man auch für Proteinsequenzen mehr Informationen erhalten und diese in einem Alignment berücksichtigen. Diese Informationen werden teilweise schon eingesetzt, wie etwa die Sekundärstruktur von Proteinen. Diese kann entweder aus Datenbanken für bereits bekannte Sequenzen oder Bereiche entnommen werden und gibt dann Versuchsergebnisse wieder, oder wird von Heuristiken vorhergesagt. Bei solchen vorhergesagten Sekundärstrukturen treten allerdings noch sehr große Unsicherheiten auf, die bei der Verwendung dieser Informationen berücksichtigt werden müssen. Ähnlich könnte man bei der Bestimmung eines Alignments auf Datenbanken zurückgreifen, in denen bekannte Sequenzteile gespeichert sind, die nach Beobachtungen bei anderen Alignments häufig in Sequenzen eingefügt oder entfernt werden und so zu typischen Deletionen oder Insertionen führen.

Ein Problem aller Alignmentverfahren sind nicht konservierte Bereiche, für die kein gutes Alignment existiert. Verfahren sollten dann in den anderen Berei-

chen trotzdem ein gutes Alignment liefern, um die vorhandenen Informationen möglichst gut auszuwerten.

4.4 Probleme bei SPScore und Gap-Penalty

Neben den grundsätzlicheren Problemen der vorangehenden Abschnitte gibt es Probleme in den häufig verwendeten Bewertungssystemen. So liefert der SPScore mit einem geeigneten Scoring-System ein relativ gutes Gesamtbild von der Qualität des Multiple Alignment, liefert aber bei kleinen lokalen Fehlern schon große Bestrafungen. In Abbildung 4.1 ist ein Beispiel von 5 Sequenzen dargestellt, bei denen eine Spalte ohne Matches bei entsprechendem Scoring-System niedriger bestraft wird, als drei Mismatches, die verteilt auftreten.

·····	A	···	A	···	A	·····		A	···
·····	G	···	A	···	A	·····		E	···
·····	A	···	E	···	A	·····		G	···
·····	A	···	A	···	E	·····		R	···
·····	A	···	A	···	A	·····		Q	···
Anzahl Mismatches :	4	+	4	+	4	= 12		10	

Abbildung 4.1: Beispiel: SPScore von 5 Sequenzen

Dieser Effekt kann auf mehrere Arten abgeschwächt werden: Beispielsweise wäre eine Gewichtung mit Hilfe von bekannten Verwandtschaftsverhältnissen denkbar, so dass eine weit entfernt verwandte Sequenz keinen so großen Mismatch-Beitrag zum Score mehr liefert. Dabei könnten wie im Kapitel 3 dieser Arbeit bereits bekannte Proteinfamilieninformationen berücksichtigt werden. Eine andere Herangehensweise an dieses Problem stellt etwa [GKB00] dar, bei denen nicht alle paarweisen Scores zwischen den Sequenzen addiert werden. Dort wird mit einem vorher bestimmten phylogenetischen Baum eine Reihenfolge der Sequenzen berechnet und nur die paarweisen Scores der in dieser Reihenfolge aufeinander folgenden Sequenzen addiert. Dadurch ist jede Sequenz nur zweifach am Score beteiligt und nicht $(k-1)$-fach wie beim SPScore.

Ein Problem, das bei paarweisen Alignments und damit auch in vielen Verfahren auftritt, ist die geeignete Gap-Penalty. Aus biologischer Sicht steht ein Gap beliebiger Größe für eine Insertion oder Deletion und damit genau ein evolutionäres Ereignis. Dadurch wird eine Bevorzugung eines großen Gaps vor vielen kleinen Gaps der gleichen Gesamtlänge klar, da diese für mehrere Ereignisse stehen, bei der die gleiche Sequenz entsteht wie bei dem einen Ereignis, für das der lange Gap steht. Daher ist es angebracht, über Methoden zur Zusammenfassung von Gaps nachzudenken, wie etwa in [KG99].

4.5 Formalisierung

Einen besseren Ansatz für die Beschreibung evolutionärer Ereignisse dürfte der Übergang zu Phylogenien und die Bewertung von Alignments anhand von Phylogenien, beispielsweise mit der Maximum-Likelyhood-Methode, bieten. Diese schon in Abschnitt 4.1 beschriebene Vorgehensweise wird auch schon in der Praxis angewendet, wobei jedoch noch eine genaue Betrachtung der Bewertung mit allen Auswirkungen fehlt. Interessant ist an dieser Stelle, dass trotzdem noch das Multiple Alignment als Zwischenschritt zur Bestimmung der Tree Alignments verwendet wird. Offenbar eignet sich Multiple Alignment wegen der Zuordnung von Bereichen in Sequenzen besser zur Identifikation von Gemeinsamkeiten.

Zur Zeit werden Multiple Alignment etwa durch die in Kapitel 3 beschriebenen Verfahren bestimmt, dann aber noch einer Nachbearbeitung unterzogen. Diese kann durch eine Verbesserung durch die oben erwähnte Maximum-Likelyhood-Bewertung erfolgen, oder durch Expertenwissen. Ein wichtiger Schritt zur automatischen Erstellung biologisch guter Alignments, die direkt Aussagen über Funktionen oder Phylogenien bei Proteinen bieten können, ist, dass eine Formalisierung der Kriterien der Nachbearbeitung erfolgt. Dazu muss also ein Modell gebildet werden, das auch das derzeit angewendete Expertenwissen möglichst gut einbezieht.

Anhang A

Beschreibung des Programms MScore

MScore ist nach einem objektorientierten Ansatz aufgebaut, der es ermöglicht, die verschiedenen Bewertungsfunktionen für Alignments zu kombinieren. Mit dem folgenden Überblick über die Klassen im Programm ist es vielleicht möglich, die danach beschriebenen Kommandozeilenparameter besser zu verstehen und eventuell eigene Erweiterungen des Programms vorzunehmen.

A.1 Klassen in MScore

Die Klasse `Alignment` dient zur Speicherung eines Alignments und der Sekundärstruktur, falls diese bekannt ist. Die Sequenzen des Alignments werden im Feld `sequences` abgelegt, das Einträge vom Typ `Alchar` enthält. `Alchar` enthält jeweils ein Zeichen und eine dazugehörende Sekundärstruktur-Information `secstr`, in der die Art der Struktur sowie deren Anfangs- und Endpositionen abgelegt sind. Die Klasse `Alignment` kann mit einem GCG/MSF-File initialisiert werden. zusätzlich können noch ein Feature-Table-File und ein dazu gehörendes Referenzalignment in Form eines GCG/MSF-Files angegeben werden. Damit werden die im Feature-Table-File enthaltenen Positionsdaten des Referenzalignments auf das aktuelle Alignment umgerechnet und eingetragen.

Zur Bewertung steht die abstrakte `Score`-Klasse zur Verfügung, in deren Subklassen verschiedene Arten von Scores implementiert werden. Bis jetzt gibt es zwei Subklassen: `DiffScore` und `SPScore`. In `DiffScore` sind die Differenz-Scores implementiert, weshalb bei der Initialisierung ein zu bewertendes Alignment und ein Referenzalignment angegeben werden müssen. Die Klasse `SPScore` bietet eine Bewertung nach Scorematrizen, weshalb als Parameter ein `Alignment` und eine `ScoreMatrix` angegeben werden müssen. Bei beiden beschriebenen Klassen können zusätzlich noch eine Sekundärstrukturgewichtung `SecWeight` und eine Familiengewichtung `PwDistWeight` angegeben werden, die weiter unten beschrieben sind.

Die Klasse `ScoreMatrix` ist eine abstrakte Klasse, deren Subklassen beliebige Scoring-Systeme darstellen, wie etwa `PredefScore`, die die Verwendung bestimmter vordefinierter Systeme, wie BLOSUM oder PAM, ermöglicht.

Die Subklasse `IndepSecW` von `SecWeight` bietet die Möglichkeit, für beide behandelten Sekundärstrukturarten zwei unabhängige Funktionen zu verwenden, wie sie in den Subklassen von `SecWeightFunc` oder Subklassen implementiert sind. Diese lassen sich in zwei Arten unterteilen: `BlockScaleFixed` und `BlockScaleProp` liefern über den gesamten betrachteten Bereich den angegebenen Skalierungsfaktor, wobei dieser dem Sekundärstrukturbereich entspricht, bei dem eventuell an den Enden eine feste (`BlockScaleFixed`) oder von der Länge des Bereichs abhängige (`BlockScaleProp`) Anzahl Zeichen der Struktur ignoriert werden. Bei `LinearFixed` und `LinearProp` werden hingegen die Bereiche am Ende nicht einfach ignoriert, sondern eine stückweise lineare Funktion verwendet.

Als Proteinfamiliengewichtung `PwDistWeight` gibt es die Subklassen `PwDistDynPrg` und `GroupPwCut`. Die erste erlaubt eine Gewichtung mit den paarweisen Sequenz-Ähnlichkeiten, die mit einem angegebenen Scoring-System mit dem Needleman-Wunsch-Algorithmus bestimmt werden. Diese können zusätzlich noch skaliert werden oder in paarweise Distanzen umgerechnet werden. `GroupPwCut` erlaubt die Gewichtung nach bekannten Proteinfamilien bzw. -gruppen, die aus einem BAliBASE-File eingelesen werden können. Neben der Vorgabe-Einstellung, die zwischen den Gruppen mit 0 gewichtet und innerhalb der Gruppen mit 1, können beliebige andere Werte angegeben werden.

Bei `SecWeight` und `PwDistWeight` gibt es jeweils noch eine Default-Klasse, die keine Gewichtung vornimmt und für die ungewichteten Fälle Verwendung findet.

A.2 Kommandozeilenparameter

In der Tabelle A.1 sind die Kommandozeilenparameter des Programms MScore zu finden. Die Aufrufsyntax des Programms lautet:

```
mscore [Options] alignmentfile.msf
```

Als Beispiele sind hier die Optionen angegeben, wie sie für den Vergleich in Kapitel 3 eingesetzt wurden. Dabei werden für Variablen, gekennzeichnet durch das Zeichen `$` bestimmte Filenamen eingesetzt, deren Bedeutung in Tabelle A.2 erklärt ist.

- **dis** (Distanz-Score):

  ```
  mscore -d -r $reffile $file
  ```

- **sdis** (Distanz-Score mit Sekundärstrukturgewichtung):

  ```
  mscore -d -f $ftbfile -r $reffile -s 2 -t 2 $file
  ```

Option	Beschreibung
`-h`	Anzeige einer Übersicht der Optionen
`-a`	Ausgabe der Alignment-Parameter
`-f secfile.ftb`	Angabe des Feature-Table-Files (Sekundärstrukturen)
`-r reffile.msf`	Referenzalignment für Distanzscore und Option `-f`
`-d`	Distanz-Score berechnen
`-s secscale`	Skalierung für α-Helix
`-S secoffs`	Offset für α-Helix-Gewichtung. `secoffs` < 1 für anteiligen und `secoffs` ≥ 1 für absoluten Offset
`-l`	stückweise lineare Gewichtung für α-Helix verwenden
`-t secscale`	Skalierung für β-Faltblatt
`-T secoffs`	Offset für β-Faltblatt. `secoffs` < 1 für anteiligen und `secoffs` ≥ 1 für absoluten Offset
`-L`	stückweise lineare Gewichtung für β-Faltblatt verwenden
`-m matrname`	Berechne Score mit Scoring-System `matrname`
`-M`	Anzeige aller gültigen Scoring-Systeme
`-g gappenalty`	Setzen einer Gap-Penalty (ansonsten Durchschnitt der Mismatches)
`-p`	Gewichtung mit paarweisen Sequenzähnlichkeiten
`-P scale`	Wie `-p`, aber skaliert auf Intervall [0 ... `scale`] Bei negativem Wert von `scale` erfolgt eine Skalierung auf [0 ... $-$`scale`] und es werden paarweise Distanzen verwendet.
`-q matrname`	Scoringsystem für `-p`, falls von durch `-m` angegebenen verschieden
`-Q gappenalty`	Gap-Penalty zu `-q`
`-G familyfile`	Gewichtung mit Proteinfamilien, aus Datei `familyfile`
`-o scale`	Gewicht für Scores innerhalb der Proteinfamilie
`-O scale`	Gewicht für Scores zwischen Proteinfamilien

Tabelle A.1: Kommandozeilenoptionen von MScore

Name	Bedeutung
`$file`	Zu bewertendes Alignment
`$ftbfile`	Sekundärstruktur-Info (BAliBASE)
`$reffile`	Referenzalignment (BAliBASE)
`$groupfile`	Proteinfamilien-Info (BAliBASE)

Tabelle A.2: Filename-Variablen in den Programmaufrufen

- **csdis** (**sdis**-Score nur innerhalb von Proteinfamilien):

```
mscore -d -f $ftbfile -r $reffile -s 2 -t 2 -G $groupfile
$file
```

- **blosum** (BLOSUM62-Score):

```
mscore -m blosum62 $file
```

- **sblosum** (BLOSUM62-Score mit Sekundärstrukturgewichtung):

```
mscore -m blosum62 -f $ftbfile -r $reffile -s 2 -t 2 $file
```

- **csblosum** (**sblosum**-Score nur innerhalb von Proteinfamilien):

```
mscore -m blosum62 -f $ftbfile -r $reffile -s 2 -t 2
-G $groupfile $file
```

Anhang B

Inhalt der CD

Pfad / Verzeichnisname			Beschreibung
`LaTeX`			LaTeX-Quellen dieser Arbeit
`LaTeX-ext`			Benötigte Ergänzungen zu LaTeX
`paper`			Literatur (Auswahl)
`mscore-1_7-src`			Quellcode des Programms MScore
`multal/`			Alle Dateien des Vergleichs
	`refresh_data`		Skript: Kopieren der Testdatensätze
	`mscore`		Verzeichnis mit Binärdatei von MScore
	`filter`		Diverse Konvertierungsskripte
	`alcompar/`		Untersuchte Programme und Daten
		`BAliBASE`	Verzeichnisse mit den Testalignments
		`alignments`	Verzeichnisse mit den Testergebnissen
		`gnuplots`	Testergebnisse als Gnuplot-Graphiken
		`msa`	Quelltext des Programms MSA
		⋮	Quelltexte der anderen Programme

Tabelle B.1: Inhalt der CD

Danksagung

An dieser Stelle möchte ich mich bei denen bedanken, die diese Arbeit möglich gemacht haben. Mein besonderer Dank gilt meinen Eltern, ohne die das Studium und die Anfertigung dieser Diplomarbeit nicht möglich gewesen wäre. Meinem Betreuer Herrn Prof. Clemens Lautemann danke ich für die gute und geduldige Betreuung mit vielen Hinweisen und Vorschlägen und nicht zuletzt für die Themenstellung, die mir die Beschäftigung mit dem interdisziplinären Themenbereich Bio-Informatik ermöglicht hat. Bei Herrn Dr. Fred Sobik möchte ich mich für viele anregende Diskussionen zu verschiedensten Themen der Bio-Infomatik bedanken und für die Geduld und Anregungen beim Korrekturlesen. Bei Constanze Lipowsky möchte ich mich für das Korrekturlesen und die ausführliche Beantwortung meiner Fragen zu Themen der Molekulargenetik bedanken.

Weiter möchte ich mich bei Mitarbeitern verschiedener Arbeitsgruppen in der Biologie danken, die mir halfen, den Einsatz und die Probleme des Multiple Sequence Alignments in der Praxis nachzuvollziehen: Herr Prof. Erwin Schmidt und Dr. Thomas Hankeln vom Institut für Molekulargenetik, gentechnische Sicherheitsforschung und Beratung, sowie Herr PD Dr. Thorsten Burmester vom Institut für Zoologie und Herr Thorsten Schweikardt vom Institut für Molekulare Biophysik der Johannes Gutenberg-Universität Mainz.

Literaturverzeichnis

[ACL89] S. F. Altschul, R. J. Carroll, and D. J. Lipman. Weights for data related by a tree. *J. Mol. Biol.*, 207:647–653, 1989.

[BCL87] D. Bashford, C. Chothia, and A. Lesk. Determinants of a protein fold: unique features of the globin amino acid sequences. *J. Mol. Biol.*, 196:199–216, 1987.

[Bei00] E. Beitz. TEXshade: shading and labeling multiple sequence alignments using LATEX 2_ε. *Bioinformatics*, 16:135–139, 2000.

[BLP94] V. Bafna, E.L. Lawler, and P.A. Pevzner. Approximation algorithms for multiple sequence alignment. In *Proc. of the 5 th Symp. on Combinatorial Pattern Matching*, volume LNCS 807, pages 43–53, 1994.

[BTTP] A. Bahr, J. Thompson, J.-C. Thierry, and O. Poch. Balibase (version 2.0): A benchmark alignment database, including enhancements for repeats, transmembrane sequences and circular permutations. http://www-igbmc.u-strasbg.fr/BioInfo/BAliBASE2/index.html.

[CB00] P. Clote and R. Backofen. *Computational Molecular Biology.* John Wiley and Sons, 2000.

[CL88] H. Carillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48:1073–1082, 1988.

[CWM] J. Cooper, J. Walshaw, and A. Mills. Principles of protein structure, comparative protein modelling and visualisation. http://www.expasy.ch/swissmod/course/course-index.htm.

[DSO79] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Structure*, volume 5(Suppl. 3), pages 345–352. National Biomedical Reasearch Foundataion, Silver Spring, Md., 1979.

[Edd98] S.R. Eddy. Profile hidden markov models. *Bioinformatics*, 14 (9):755–763, 1998.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability.* Freeman, 1979.

[GKB00] G. Gonnet, C. Korostensky, and S. Benner. Evaluation measures of multiple sequence alignments. *J. Comput. Biol.*, 7(1-2):261–276, 2000.

[GKS95] S. Gupta, J. Kececioglu, and A. Schaffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comput. Biol.*, 2 (3):459–472, 1995.

[Got96] O. Gotoh. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J. Mol. Biol.*, 264:823–838, 1996.

[Gus97] D. Gusfield. *Algorithms on Strings, Tress and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, 1997.

[HH92] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Academy Science*, 89 (10):915–919, 1992.

[HS96] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[JL94] T. Jiang and M. Li. Optimization problems in molecular biology. In D.Z.Du and J. Sun, editors, *Advances in Optimization and Approximation*, pages 195–216. Kluwer Academic Publishers, MA, 1994.

[Jus99] W. Just. Computational complexity of multiple sequence alignment with sp-score. *Technical report Ohio University*, 1999.

[KBM+93] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modeling. Technical Report UCSC-CRL-93-32, 1993.

[KG99] C. Korostensky and G. Gonnet. Gap heuristics and tree construction using gaps. Technical Report 321, 1999.

[LAK89] D. J. Lipman, S. F. Altschul, and J. D. Keceioglu. A tool for multiple sequence alignment. In *Proc. Natl. Acad. Sci. USA*, volume 86, pages 4412–4415, 1989.

[LG00] W. Li and D. Graur. *Fundamentals of Molecular Evolution.* Sinauer Associates, Inc., second edition, 2000.

[LHM+97] R.A. Laskowski, E.G. Hutchinson, A.D. Michie, A.C. Wallace, M.L. Jones, and J.M. Thornton. Pdbsum: a web-based database of summaries and analyses of all pdb structures. *Trends Biochem. Sci.*, 22(12):488–490, 1997.

[MDBO98] K. Mizuguchi, C.M. Deane, T.L. Blundell, and J.P. Overington. Homstrad: a database of protein structure alignments for homologous families. *Protein Sci.*, 7:2469–2471, 1998.

[MDW96] B. Morgenstern, A. Dress, and T. Werner. Multiple dna and protein sequence alignment based on segment-to-segment comparison. In *Proc. Natl. Acad. Sci. USA*, volume 93, pages 12098–12103, 1996.

[MGB95] T. Madej, J. Gibrat, and S. Bryant. Threading a database of protein cores. *Proteins: Struct. Funct. Genet.*, 23:356–369, 1995.

[Mor99] B. Morgenstern. Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15 (3):211–218, 1999.

[NHH00] C. Notredame, D.G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, 302:205–217, 2000.

[NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to the search of similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, New York, 1994.

[PCB51] L. Pauling, R. Corey, and H. Branson. The structure of proteins: Two hydrogen-bonded helical configurations of the polypeptide chain. *Proc. Natl. Acad. Sci. USA*, 37:205–210, 1951.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *J. Comp. Sys. Sci.*, 43:425–440, 1991.

[Rab89] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, (77 , no. 2):257–285, 1989.

[Sch99] T. Schwentick. Skript zur Vorlesung Grundzüge der Informatik II. ftp://ftp.informatik.uni-mainz.de/pub/sources/gdi/skript99.ps, 1999.

[SdDS97] R. Schneider, A. de Daruvar, and C. Sander. The hssp database of protein structure-sequence alignments. *Nucleic Acids Res.*, 25:226–230, 1997.

[SKM01] L. Shvartser, C. Kulikowski, and I. Muchnik. Multiple sequence alignment using the quasi-concave function optimization based on the dialign combinatorial structures. Technical Report 2001-02, DIMACS, 2001.

[SM97] J. Setubal and J. Mendianis. *Introduction to Computational Molecular Biology.* PWS Pub. Co., 1997.

[SN87] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. and Evol.*, 4:406–425, 1987.

[SW92] E. Sweedyk and T. Warnow. The optimal tree alignment problem is np-hard. *manuscript*, 1992.

[Tay99] W. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Sci.*, 8:654–665, 1999.

[THG94] J. Thompson, D. Higgins, and T. Gibson. Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[TPP99a] J. Thompson, F. Plewniak, and O. Poch. Balibase: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15 (1):87–88, 1999.

[TPP99b] J. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27 (13):2682–2690, 1999.

[WG96] L. Wang and D. Gusfield. Improved approximation algorithms for tree alignment. In D. S. Hirschberg and E. W. Myers, editors, *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, number 1075, pages 220–233, Laguna Beach, CA, 1996. Springer-Verlag, Berlin.

[WJ94] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Bio.*, 1:337–348, 1994.

[WJG00] L. Wang, T. Jiang, and D. Gusfield. A more efficient approximation scheme for tree alignment. *SIAM J. Comput.*, 30(1):283–299, 2000.

[Yan78] M. Yannakakis. Node- and edge-deletion NP-complete problems. In *Proc. 10th Ann. ACM Symp. on Theory of Computing*, pages 253–264. Association for Computing Machinery, New York, 1978.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Mainz, den 1. Februar 2002

Götz Schwandtner