INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR EFFIZIENTE ALGORITHMEN

Skriptum

zur Vorlesung

Algorithmische Bioinformatik I/II

gehalten im Wintersemester 2001/2002

und im Sommersemester 2002 von

Volker Heun

Erstellt unter Mithilfe von:

Peter Lücke – Hamed Behrouzi – Michael Engelhardt

Sabine Spreer – Hanjo Täubig

Jens Ernst – Moritz Maaß

14. Mai 2003

Version 0.96

Vorwort

Dieses Skript entstand parallel zu den Vorlesungen *Algorithmische Bioinformatik I* und *Algorithmische Bioinformatik II*, die im Wintersemester 2001/2002 sowie im Sommersemester 2002 für Studenten der Bioinformatik und Informatik sowie anderer Fachrichtungen an der Technischen Universität München im Rahmen des von der Ludwig-Maximilians-Universität und der Technischen Universität gemeinsam veranstalteten Studiengangs Bioinformatik gehalten wurde. Einige Teile des Skripts basieren auf der bereits im Sommersemester 2000 an der Technischen Universität München gehaltenen Vorlesung *Algorithmen der Bioinformatik* für Studierende der Informatik.

Das Skript selbst umfasst im Wesentlichen die grundlegenden Themen, die man im Bereich Algorithmische Bioinformatik einmal gehört haben sollte. Die vorliegende Version bedarf allerdings noch einer Ergänzung weiterer wichtiger Themen, die leider nicht in den Vorlesungen behandelt werden konnten.

An dieser Stelle möchte ich insbesondere Hamed Behrouzi, Michael Engelhardt und Peter Lücke danken, die an der Erstellung des ersten Teils dieses Skriptes (Kapitel 2 mit 5) maßgeblich beteiligt waren. Bei Sabine Spreer möchte ich mich für die Unterstützung bei Teilen des siebten Kapitels bedanken. Bei meinen Übungsleitern Jens Ernst und Moritz Maaß für deren Unterstützung der Durchführung des Übungsbetriebs, aus der einige Lösungen von Übungsaufgaben in dieses Text eingeflossen sind. Bei Hanjo Täubig möchte ich mich für die Mithilfe zur Fehlerfindung bedanken, insbesondere bei den biologischen Grundlagen.

Falls sich dennoch weitere (Tipp)Fehler unserer Aufmerksamkeit entzogen haben sollten, so bin ich für jeden Hinweis darauf (an heun@in.tum.de) dankbar.

München, im September 2002 — Volker Heun

Inhaltsverzeichnis

Paarweises Sequenzen Alignment

In diesem Kapitel beschäftigen wir uns mit dem so genannten „Paarweisen Sequenzen Alignment“. Dabei werden zwei Sequenzen bzw. Zeichenreihen miteinander verglichen und darauf untersucht, wie „ähnlich“ sie sind und wie die eine Sequenz aus der anderen hervorgegangen sein kann. Dies wird an folgenden Beispielen illustriert:

Betrachte die Worte Monkey und Money:

Insertion

M o n k e y

M o n _ e y

Deletion

Wie obiges Bild bereits zeigt, kann das Wort Money aus dem Wort Monkey dadurch hervorgegangen sein, dass im Wort Monkey einfach das „k“ gelöscht wurde. Andersherum kann in das Wort Money ein „k“ eingefügt worden sein, so dass das Wort Monkey entstand.

Ähnlich verhält es sich mit den Wörtern Money und Honey, wie folgendes Bild zeigt:

M o n e y

Substitution

H o n e y

Hier wurde entweder ein „M“ durch ein „H“ oder ein „H“ durch ein „M“ ersetzt.

Um nun tiefer in das Thema einsteigen zu können, sind zuerst einige grundlegende Definitionen nötig.

3.1 Distanz- und Ähnlichkeitsmaße

In diesem Abschnitt werden wir so genannte Distanzmaße einführen, die es uns überhaupt erst erlauben, ähnliche Sequenzen qualitativ und quantitativ zu beurteilen.

3.1.1 Edit-Distanz

Sei hier und im Folgenden Σ ein Alphabet. Weiter sei „$-$" ein neues Zeichen, d.h. $- \notin \Sigma$. Dann bezeichne $\overline{\Sigma} := \Sigma \cup \{-\}$ das um $-$ erweiterte Alphabet von Σ. Wir werden $-$ oft auch als *Leerzeichen* bezeichnen.

Definition 3.1 *Eine* Edit-Operation *ist ein Paar*

$$(x, y) \in \overline{\Sigma} \times \overline{\Sigma} \setminus \{-, -\}.$$

Eine Edit-Operation (x, y) heißt

- Match*, wenn $x = y \in \Sigma$;*
- Substitution*, wenn $x \neq y \in \Sigma$;*
- Insertion*, wenn $x = -, y \in \Sigma$;*
- Deletion*, wenn $x \in \Sigma, y = -$.*

Wir bemerken hier, dass $(x, x) \in \Sigma \times \Sigma$ explizit als Edit-Operation zugelassen wird. In vielen Büchern wird dies nicht erlaubt. Wir werden im Folgenden sehen, dass ein solcher Match als Edit-Operation eigentlich eine neutrale oder NoOp-Operation ist und meist problemlos weggelassen werden kann. Als *Indel-Operation* bezeichnet man eine Edit-Operation, die entweder eine Insertion oder Deletion ist.

Definition 3.2 *Ist (x, y) eine Edit-Operation und sind $a, b \in \Sigma^*$, dann gilt $a \xrightarrow{(x,y)} b$, d.h. a kann mit Hilfe der Edit-Operation (x, y) in b umgeformt werden,*

- *wenn $\exists\, i \in [1 : |a|] : (x, y \in \Sigma) \wedge (a_i = x) \wedge (b = a_1 \cdots a_{i-1} \cdot y \cdot a_{i+1} \cdots a_{|a|})$ (Substitution oder Match);*
- *oder wenn $\exists\, i \in [1 : |a|] : (a_i = x) \wedge (y = -) \wedge (b = a_1 \cdots a_{i-1} \cdot a_{i+1} \cdots a_{|a|})$ (Deletion);*
- *oder wenn $\exists\, j \in [1 : |b|] : (b_j = y) \wedge (x = -) \wedge (a = b_1 \cdots b_{j-1} \cdot b_{j+1} \cdots b_{|b|})$ (Insertion).*

Sei $s = ((x_1, y_1), \ldots, (x_m, y_m))$ eine Folge von Edit-Operationen mit $a_{i-1} \xrightarrow{(x_i,y_i)} a_i$, wobei $a_i \in \Sigma^*$ für $i \in [0 : m]$ und $a := a_0$ und $b := a_m$, dann schreibt man auch kurz $a \stackrel{s}{\Rightarrow} b$.

Folgende Abbildungen zeigen zwei Beispiele von Transformationen einer Sequenz in eine andere mit Hilfe von Edit-Operationen. Dabei wird immer die obere Sequenz in die untere umgewandelt.

$$AGTGTAGTA \overset{s}{\Rightarrow} ACGTGTTT \text{ mit } s = ((G,-),(T,C),(A,G),(G,T),(A,T))$$
$$\text{oder mit } s = ((G,T),(A,G),(G,-),(A,T),(T,C)).$$

	A	G	T	G	T	A	G	T	A	D: Deletion
	A	_	C	G	T	G	T	T	T	I: Insertion
5 Edit-Op.		D	S			S	S		S	S: Substitution

Abbildung 3.1: Beispiel: Transformation mit Edit-Operationen

Wie man im obigen Beispiel sieht, kommt es nicht unbedingt auf die Reihenfolge der Edit-Operationen an. Wir wollen hier nur anmerken, dass es durchaus Sequenzen von Edit-Operationen gibt, so dass es durchaus auf die Reihenfolge ankommt. Der Leser möge sich solche Beispiele überlegen.

Wir können dieselben Sequenzen auch mit Hilfe anderer Edit-Operationen ineinander transformieren.

$$AGTGTAGTA \overset{s'}{\Rightarrow} ACGTGTTT \text{ mit } s' = ((-,C),(A,-),(G,-),(A,T)).$$

	A	_	G	T	G	T	A	G	T	A	D: Deletion
	A	C	G	T	G	T	_	_	T	T	I: Insertion
4 Edit-Op.		I					D	D		S	S: Substitution

Abbildung 3.2: Beispiel: Transformation mit anderen Edit-Operationen

Um nun die Kosten einer solchen Folge von Edit-Operationen zu bewerten, müssen wir zunächst die Kosten einer einzelnen Edit-Operation bewerten. Dazu verwenden wir eine so genannte (Kosten-)Funktion $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$. Beispielsweise kann man alle Kosten bis auf die Matches auf 1 setzen; Für Matches sind Kosten größer als Null in der Regel nicht sonderlich sinnvoll. In der Biologie, wenn die Sequenzen Basen oder insbesondere Aminosäuren repräsentieren, wird man jedoch intelligentere Kostenfunktionen wählen.

Definition 3.3 *Sei* $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ *eine (Kosten-)Funktion. Seien* $a, b \in \Sigma^*$ *und sei* $s = (s_1, \ldots, s_l)$ *eine Folge von Edit-Operationen mit* $a \xrightarrow{s_1} \cdots \xrightarrow{s_l} b$ *(kurz* $a \stackrel{s}{\Longrightarrow} b$*). Dann sind die* Kosten der Edit-Operationen s *definiert als*

$$w(s) := \sum_{j=1}^{l} w(s_j).$$

Die Edit-Distanz *von* $a, b \in \Sigma^*$ *ist definiert als*

$$d_w(a, b) := min\{w(s) \mid a \stackrel{s}{\Longrightarrow} b\}.$$

Zuerst einmal überlegen wir uns, was für eine Kostenfunktion w sinnvollerweise gelten soll:

$\boldsymbol{w(x, y) + w(y, z) \geq w(x, z)}$ **für** $\boldsymbol{x, y, z \in \Sigma}$**:** Wir betrachten zum Beispiel eine Mutation (x, z), die als direkte Mutation relativ selten (also teuer) ist, sich jedoch sehr leicht (d.h. billig) durch zwei Mutationen (x, y) und (y, z) ersetzen lässt. Dann sollten die Kosten für diese Mutation durch die beiden billigen beschrieben werden, da man in der Regel nicht feststellen kann, ob eine beobachtete Mutation direkt oder über einen Umweg erzielt worden ist.

$\boldsymbol{w(x, -) + w(-, z) \geq w(x, z)}$ **für** $\boldsymbol{x, z \in \Sigma}$**:** Es ist wohl im Allgemeinen wahrscheinlicher, dass eine Mutation durch eine Substitution und nicht erst durch Deletion und eine anschließende Insertion (oder umgekehrt) zustande gekommen ist. Daher sollte die Kosten der Substitution billiger als die Kosten der entsprechenden Deletion und Insertion sein.

Diese Bedingungen sind beispielsweise erfüllt, wenn w eine Metrik ist.

Definition 3.4 *Eine Funktion* $w : M \times M \to \mathbb{R}_+$ *heißt* Metrik*, wenn die folgenden Bedingungen erfüllt sind:*

(M1) $\forall x, y \in M : w(x, y) = 0 \Leftrightarrow x = y$ *(Definitheit);*

(M2) $\forall x, y \in M : w(x, y) = w(y, x)$ *(Symmetrie);*

(M3) $\forall x, y, z \in M : w(x, z) \leq w(x, y) + w(y, z)$ *(Dreiecksungleichung).*

Oft nimmt man daher für eine Kostenfunktion für ein Distanzmaß an, dass es sich um eine Metrik handelt. Wir müssen uns nur noch M1 und M2 im biologischen Zusammenhang klar machen. M1 sollte gelten, da nur ein Zeichen mit sich selbst identisch ist und somit nur Zeichen mit sich selbst einen Abstand von 0 haben sollten. M2 ist nicht ganz so klar, da Mutationen in die eine Richtung durchaus wahrscheinlicher sein können als in die umgekehrte Richtung. Da wir allerdings immer nur die Sequenzen sehen und oft nicht die Kenntnis haben, welche Sequenz aus welcher Sequenz durch Mutation entstanden ist, ist die Annahme der Symmetrie bei Kostenfunktionen sinnvoll. Des Weiteren kommt es bei Vergleichen von Sequenzen oft vor, dass beide von einer dritten Elter-Sequenz abstammen und somit aus dieser durch Mutationen entstanden sind. Damit ist die Richtung von Mutationen in den beobachteten Sequenzen völlig unklar und die Annahme der Symmetrie der einzig gangbare Ausweg.

Lemma 3.5 *Ist $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine Metrik, dann ist auch $d_w : \overline{\Sigma}^* \times \overline{\Sigma}^* \to \mathbb{R}_+$ eine Metrik.*

Beweis: Seien $a, b \in \Sigma^*$. Dann gilt:

$$\begin{aligned} 0 &= d_w(a,b) \\ &= \min\left\{ w(s) \ : \ a \stackrel{s}{\Rightarrow} b \right\} \\ &= \min\left\{ \sum_{i=1}^{r} w(s_i) \ : \ a \stackrel{s}{\Longrightarrow} b \text{ mit } s = (s_1, \ldots, s_r) \right\}. \end{aligned}$$

Da w immer nichtnegativ ist, muss $w(s_i) = 0$ für alle $i \in [1 : r]$ sein (bzw. $r = 0$). Somit sind alle ausgeführten Edit-Operationen Matches, d.h. $s_i = (x_i, x_i)$ für ein $x_i \in \Sigma$. Damit gilt $a = b$ und somit M1 für d_w.

Seien $a, b \in \Sigma^*$. Dann gilt:

$$\begin{aligned} d_w(a,b) &= \min\left\{ w(s) \ : \ a \stackrel{s}{\Rightarrow} b \text{ mit } s = (s_1, \ldots, s_r) \right\} \\ &= \min\left\{ \sum_{i=1}^{r} w(s_i) \ : \ a = a_0 \stackrel{s_1}{\to} a_1 \cdots a_{r-1} \stackrel{s_r}{\to} a_r = b \right\} \\ &= \min\left\{ \sum_{i=1}^{r} w(s_i) \ : \ b = a_r \stackrel{\tilde{s}_r}{\to} a_{r-1} \cdots a_1 \stackrel{\tilde{s}_1}{\to} a_0 = a \right\} \\ &= \min\left\{ w(\tilde{s}^R) \ : \ a \stackrel{\tilde{s}^R}{\Rightarrow} b \text{ mit } \tilde{s}^R = (\tilde{s}_1, \ldots, \tilde{s}_r) \right\} \\ &= d_w(b,a). \end{aligned}$$

Hierbei bezeichnet $\tilde{s}_i$ für eine Edit-Operation $s_i = (x, y)$ mit $x, y \in \overline{\Sigma}$ die inverse Edit-Operation $\tilde{s}_i = (y, x)$. Damit ist auch M2 für d_w nachgewiesen.

Seien $a, b, c \in \Sigma^*$. Seien s und t zwei Folgen von Edit-Operationen, so dass $a \stackrel{s}{\Rightarrow} b$ und $w(s) = d_w(a, b)$ sowie $b \stackrel{t}{\Rightarrow} c$ und $w(t) = d_w(b, c)$. Dann ist offensichtlich die Konkatenation $s \cdot t$ eine Folge von Edit-Operationen mit $a \stackrel{s \cdot t}{\Longrightarrow} c$. Dann gilt weiter

$$d_w(a, c) \leq w(s \cdot t) = w(s) + w(t) = d_s(a, b) + d_w(b, c).$$

Damit ist auch die Dreiecksungleichung bewiesen. ■

3.1.2 Alignment-Distanz

In diesem Abschnitt wollen wir einen weiteren Begriff einer Distanz einzuführen, die auf Ausrichtungen der beiden bezeichneten Zeichenreihen beruht. Dazu müssen wir erst einmal formalisieren, was wir unter einer Ausrichtung (einem Alignment) von zwei Zeichenreihen verstehen wollen.

Definition 3.6 *Sei $u \in \overline{\Sigma}^*$. Dann ist Restriktion von u auf Σ definiert als $u|_\Sigma = h(u)$, wobei*

$$\begin{aligned} h(a) &= a && \text{für alle } a \in \Sigma, \\ h(-) &= \varepsilon, && \\ h(u'u'') &= h(u')h(u'') && \text{für alle } u', u'' \in \Sigma. \end{aligned}$$

Die Restriktion von $u \in \overline{\Sigma}^*$ ist also nichts anderes als das Löschen aller Zeichen $-$ aus u. Nun sind wir formal in der Lage, ein Alignment zu definieren.

Definition 3.7 *Ein Alignment ist ein Paar $(\overline{a}, \overline{b}) \in \overline{\Sigma}^*$ mit $|\overline{a}| = |\overline{b}|$ und*

$$\forall i \in [1 : |\overline{a}|] : \overline{a}_i = \overline{b}_i \Rightarrow \overline{a}_i \neq - \neq \overline{b}_i.$$

$(\overline{a}, \overline{b})$ ist ein Alignment für $a, b \in \Sigma^*$, wenn gilt:

$$\overline{a}|_\Sigma = a \text{ und } \overline{b}|_\Sigma = b.$$

Betrachten wir das folgende Beispiel in Abbildung 3.3, ein Alignment zwischen $a = AGGCATT$ und $b = AGCGCTT$. Mit Hilfe solcher Alignments lassen sich jetzt ebenfalls Distanzen zwischen Zeichenreihen definieren.

$$\begin{array}{cccccccc} A & - & G & G & C & A & T & T \\ A & G & C & G & C & - & T & T \end{array}$$

Abbildung 3.3: Beispiel: Alignment von $AGGCATT$ mit $AGCGCTT$

Definition 3.8 *Sei $\overline{w} : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine (Kosten-)Funktion. Die Kosten eines Alignments $(\overline{a}, \overline{b})$ für (a, b) sind definiert als*

$$\overline{w}(\overline{a}, \overline{b}) := \sum_{i=1}^{|\overline{a}|} \overline{w}(\overline{a}_i, \overline{b}_i).$$

Die Alignment-Distanz von $a, b \in \Sigma^*$ ist definiert als

$$\overline{d}_{\overline{w}}(a, b) := \min\{\overline{w}(\overline{a}, \overline{b}) \mid (\overline{a}, \overline{b}) \text{ ist Alignment für } a, b\}.$$

Für die Kostenfunktion gilt hier dasselbe wie für die Kostenfunktion für die Edit-Distanz. Wählt man im obigen Beispiel in Abbildung 3.3 $w(x, x) = 0$ für $x \in \Sigma$ und $w(x, y) = 1$ für $x \neq y \in \overline{\Sigma}$, dann hat das gegebene Alignment eine Distanz von 3. Es gibt jedoch ein besseres Alignment mit Alignment-Distanz 2.

3.1.3 Beziehung zwischen Edit- und Alignment-Distanz

In diesem Abschnitt wollen wir zeigen, dass Edit-Distanz und Alignment-Distanz unter bestimmten Voraussetzungen gleich sind.

Lemma 3.9 *Sei $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine Kostenfunktion und seien $a, b \in \Sigma^*$. Für jedes Alignment $(\overline{a}, \overline{b})$ von a und b gibt es eine Folge s von Edit-Operationen, so dass $a \stackrel{s}{\Longrightarrow} b$ und $w(s) = w(\overline{a}, \overline{b})$*

Beweis: Sei $(\overline{a}, \overline{b})$ ein Alignment für a und b. Dann konstruieren wir eine Folge $s = (s_1, \ldots, s_{|\overline{a}|})$ von Edit-Operationen wie folgt: $s_i = (\overline{a}_i, \overline{b}_i)$. Dann gilt offensichtlich (da $(\overline{a}, \overline{b})$ ein Alignment für a und b ist): $a \stackrel{s}{\Longrightarrow} b$. Für die Edit-Distanz erhalten wir:

$$w(s) = \sum_{i=1}^{|\overline{a}|} w(s_i) = \sum_{i=1}^{|\overline{a}|} w(\overline{a}_i, \overline{b}_i) = w(\overline{a}, \overline{b}).$$

■

Aus diesem Lemma folgt sofort das folgende Korollar, das besagt, dass die Edit-Distanz von zwei Zeichenreihen höchstens so groß ist wie die Alignment-Distanz.

Korollar 3.10 *Sei $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine Kostenfunktion, dann gilt für alle $a, b \in \overline{\Sigma}^*$:*

$$d_w(a,b) \leq \overline{d}_w(a,b).$$

Nun wollen wir die umgekehrte Richtung untersuchen.

Lemma 3.11 *Sei $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine metrische Kostenfunktion und seien $a, b \in \Sigma^*$. Für jede Folge von Edit-Operationen mit $a \stackrel{s}{\Longrightarrow} b$ gibt es ein Alignment $(\overline{a}, \overline{b})$ von a und b, so dass $w(\overline{a}, \overline{b}) \leq w(s)$.*

Beweis: Wir führen den Beweis durch Induktion über $n = |s|$.

Induktionsanfang ($n = 0$): Aus $|s| = 0$ folgt, dass $s = \varepsilon$. Also ist $a = b$ und $w(s) = 0$. Wir setzen nun $\overline{a} = a = b = \overline{b}$ und erhalten ein Alignment $(\overline{a}, \overline{b})$ für a und b mit $w(\overline{a}, \overline{b}) = 0 \leq w(s)$.

Induktionsschritt ($n \to n+1$): Sei $s = (s_1, \ldots, s_n, s_{n+1})$ eine Folge von Edit-Operationen mit $a \stackrel{s}{\Longrightarrow} b$. Sei nun $s' = (s_1, \ldots, s_n)$ und $a \stackrel{s'}{\Longrightarrow} c \stackrel{s_{n+1}}{\longrightarrow} b$ für ein $c \in \Sigma^*$. Aus der Induktionsvoraussetzung folgt nun, dass es ein Alignment $(\overline{a}, \overline{c})$ von a, c gibt, so dass $w(\overline{a}, \overline{c}) \leq w(s')$.

$\overline{a}$		$*$		$\overline{a}\|_\Sigma = a$
$\overline{c}$		x		$\overline{c}\|_\Sigma = c$
$\overline{b}$		y		$\overline{b}\|_\Sigma = b$

Sei $s_{n+1} = (x, y)$ eine Substitution, Match oder Deletion.

Abbildung 3.4: Skizze: zum Induktionsbeweis (Substitution)

Wir betrachten zuerst den Fall, dass die letzte Edit-Operation $s_{n+1} = (x, y)$ eine Substitution, ein Match oder eine Deletion ist, d.h. $x \in \Sigma$ und $y \in \overline{\Sigma}$. Wir können dann, wie in Abbildung 3.4 schematisch dargestellt, ein Alignment für a und b erzeugen, indem wir die Zeichenreihe $\overline{b}$ geeignet aus $\overline{c}$ unter Verwendung der Edit-

Operation(x, y) umformen. Es gilt dann:

$$
\begin{aligned}
w(\overline{a}, \overline{b}) &= w(\overline{a}, \overline{c}) \underbrace{-w(\overline{a}_i, \overline{c}_i) + w(\overline{a}_i, \overline{b}_i)}_{\leq w(\overline{b}_i, \overline{c}_i)} \\
& \text{aufgrund der Dreiecksungleichung und der Symmetrie} \\
& \text{d.h., } w(\overline{a}_i, \overline{b}_i) \leq w(\overline{a}_i, \overline{c}_i) + w(\overline{b}_i, \overline{c}_i) \\
&\leq \underbrace{w(\overline{a}, \overline{c})}_{\leq w(s')} + \underbrace{w(\overline{b}_i, \overline{c}_i)}_{= w(s_{n+1})} \\
&\leq w(s') + w(s_{n+1}) \\
&= w(s).
\end{aligned}
$$

Es bleibt noch der Fall, wenn $s_{n+1} = (-, y)$ mit $y \in \Sigma$ eine Insertion ist. Dann erweitern wir das Alignment $(\overline{a}, \overline{c})$ von a und c zu einem eigentlich „unzulässigen Alignment" $(\overline{a'}, \overline{c'})$ von a und c wie folgt. Es gibt ein $i \in [0 : |b|]$ mit $b_i = y$ und $b = c_1 \cdots c_i \cdot y \cdot c_{i+1} \cdots c_{|a|}$. Sei j die Position, nach der das Symbol y in $\overline{c}$ eingefügt wird. Dann setzen wir $\overline{a} = \overline{a}_1 \cdots \overline{a}_j \cdot - \cdot \overline{a}_{j+1} \cdots \overline{a}_{|\overline{c}|}$, $\overline{c} = \overline{c}_1 \cdots \overline{c}_j \cdot - \cdot \overline{c}_{j+1} \cdots \overline{c}_{|\overline{c}|}$ und $\overline{b} = \overline{c}_1 \cdots \overline{c}_j \cdot y \cdot \overline{c}_{j+1} \cdots \overline{c}_{|\overline{c}|}$. Dies ist in Abbildung 3.5 noch einmal schematisch

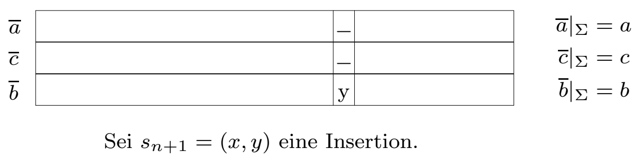


Abbildung 3.5: Skizze: zum Induktionsbeweis(Insertion)

dargestellt. $(\overline{a}, \overline{c})$ ist jetzt kein Alignment mehr, da es eine Spalte $(-, -)$ gibt. Wir interessieren uns jedoch jetzt nur noch für das Alignment $(\overline{a}, \overline{b})$ von a und b. Damit gilt

$$
\begin{aligned}
w(\overline{a}, \overline{b}) &= w(\overline{a}, \overline{c}) + w(-, y) \\
& \text{Nach Induktionsvoraussetzung} \\
&\leq w(s') + w(s_n) \\
&= w(s).
\end{aligned}
$$

■

Aus diesem Lemma folgt jetzt, dass die Alignment-Distanz durch die Edit-Distanz beschränkt ist, sofern die zugrunde liegende Kostenfunktion eine Metrik ist.

Korollar 3.12 *Ist* $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ *eine metrische Kostenfunktion, dann gilt für alle* $a, b \in \Sigma^*$*:*

$$\overline{d}_w(a, b) \leq d_w(a, b).$$

Fasst man die beiden Korollare zusammen, so ergibt sich das folgende Theorem, dass die Edit-Distanz mit der Alignment-Distanz zusammenfallen, wenn die zugrunde gelegte Kostenfunktion eine Metrik ist.

Theorem 3.13 *Ist* w *eine Metrik, dann gilt:* $d_w = \overline{d}_w$

Man überlege sich, dass alle drei Bedingungen der Metrik wirklich benötigt werden. Insbesondere für die Definitheit mache man sich klar, dass man zumindest auf die Gültigkeit von $w(x, x) = 0$ für alle $x \in \Sigma$ nicht verzichten kann.

Manchmal will man aber auf die Definitheit verzichten, d.h. manche Zeichen sollen gleicher als andere sein. Dann schwächt man die Bedingungen an die Kostenfunktion $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ wie folgt ab.

Definition 3.14 *Eine Kostenfunktion* $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ *für ein Distanzmaß heißt* sinnvoll*, wenn folgende Bedingungen erfüllt sind:*

(D1) $\max\{w(x, x) \mid x \in \Sigma\} \leq \min\{w(x, y) \mid x \neq y \in \overline{\Sigma}\}$*;*

(D2) $\forall x, y \in \overline{\Sigma} : w(x, y) = w(y, x)$ *(Symmetrie);*

(D3) $\forall x, y, z \in \overline{\Sigma} : w(x, z) \leq w(x, y) + w(y, z)$ *(Dreiecksungleichung).*

Von der Definitheit bleibt hier also nur noch übrig, dass gleiche Zeichen einen geringeren Abstand (aber nicht notwendigerweise 0) haben müssen als verschiedene Zeichen. Will man die Äquivalenz von Edit- und Alignment-Distanz weiterhin sicherstellen, so wird man ebenfalls $\max\{w(x, x) \mid x \in \Sigma\} = 0$ fordern. Im Folgenden werden wir für alle Kostenfunktionen für Distanzmaße voraussetzen, dass zumindest D1, D2 und D3 gelten.

3.1.4 Ähnlichkeitsmaße

Für manche Untersuchungen ist der Begriff der Ähnlichkeit von zwei Zeichen angemessener als der Begriff der Unterschiedlichkeit. Im letzten Abschnitt haben wir gesehen, wie wir Unterschiede zwischen zwei Zeichenreihen qualitativ und quantitativ fassen können. In diesem Abschnitt wollen wir uns mit der Ähnlichkeit von Zeichenreihen beschäftigen. Zum Ende dieses Abschnittes werden wir zeigen, dass sich die hier formalisierten Begriffe der Distanz und der Ähnlichkeit im Wesentlichen entsprechen.

Im Unterschied zu Distanzen, werden gleiche Zeichen mit einem positiven Gewicht belohnt, während ungleiche Zeichen mit einen negativen Gewicht bestraft werden. Man hat also insbesondere die Möglichkeit, die Gleichheit von gewissen Zeichen stärker zu bewerten als von anderen Zeichen. Formal lässt sich eine Kostenfunktion für Ähnlichkeitsmaße wie folgt definieren.

Definition 3.15 *Ein Kostenfunktion* $w' : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}$ *für ein Ähnlichkeitsmaß heißt* sinnvoll, *wenn die folgenden Bedingungen erfüllt sind:*

(S1) $\forall x \in \Sigma : w'(x,x) \geq 0$*;*

(S2) $\forall x \neq y \in \overline{\Sigma} : w'(x,y) \leq 0$*;*

(S3) $\forall x,y \in \overline{\Sigma} : w'(x,y) = w'(y,x)$*.*

Im Folgenden werden wir für alle Kostenfunktionen für Ähnlichkeitsmaße voraussetzen, dass S1, S2 und S3 gelten. Basierend auf solchen Kostenfunktionen für Ähnlichkeitsmaße können wir jetzt die Ähnlichkeit von Alignments definieren.

Definition 3.16 *Sei* $w' : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}$ *eine Kostenfunktion und sei* $(\overline{a}, \overline{b})$ *ein Alignment für* $a, b \in \Sigma^*$*. Dann ist die Ähnlichkeit von* $(\overline{a}, \overline{b})$ *definiert als:*

$$w'(\overline{a}, \overline{b}) := \sum_{i=1}^{|\overline{a}|} w'(\overline{a_i}, \overline{b_i})$$

Die Ähnlichkeit von $a, b \in \Sigma^*$ *ist definiert als:*

$$s(a,b) := \max\{w'(\overline{a}, \overline{b}) \mid (\overline{a}, \overline{b}) \text{ ist Alignment für } a, b\}$$

3.1.5 Beziehung zwischen Distanz- und Ähnlichkeitsmaßen

Die folgenden beiden Lemmata verdeutlichen die Beziehung, die zwischen Ähnlichkeitsmaß und Distanzmaß besteht.

Lemma 3.17 *Sei w eine sinnvolle Kostenfunktion für ein Distanzmaß. Dann existiert ein $C \in \mathbb{R}_+$, so dass $w'(a,b) = C - w(a,b)$ eine sinnvolle Kostenfunktion für ein Ähnlichkeitsmaß ist.*

Beweis: Sei $A := \max\{w(a,a) : a \in \Sigma\}$ und $B := \min\left\{w(a,b) : a \neq b \in \overline{\Sigma}\right\}$. Da w eine sinnvolle Kostenfunktion für ein Distanzmaß ist, gilt $A \leq B$. Wir wählen daher $C \in [A : B]$.

Für $a \in \Sigma$ gilt:

$$w'(a,a) = C - w(a,a) \geq A - w(a,a) = \max\{w(a,a) : a \in \Sigma\} - w(a,a) \geq 0.$$

Somit haben wir die Eigenschaft S1 nachgewiesen.

Für $a \neq b \in \overline{\Sigma}$ gilt:

$$w'(a,b) = C - w(a,b) \leq B - w(a,b) = \min\left\{w(a,b) : a \neq b \in \overline{\Sigma}\right\} - w(a,b) \leq 0.$$

Somit haben wir die Eigenschaft S2 nachgewiesen.

Die Eigenschaft S3 folgt offensichtlich aus der Definition von w' und D2. ■

Lemma 3.18 *Sei $w' : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine sinnvolle Kostenfunktion für ein Ähnlichkeitsmaß. Dann existiert ein $D \in \mathbb{R}_+$, so dass $w(a,b) = D - w'(a,b)$ eine sinnvolle Kostenfunktion für ein Distanzmaß ist.*

Beweis: Wir wählen $D = \max\left\{w'(a,b) : a,b \in \overline{\Sigma}\right\}$. Nach Wahl von D gilt dann

$$\begin{aligned} w(a,b) &= D - w'(a,b) \\ &= \max\left\{w'(a,b) : a,b \in \overline{\Sigma}\right\} - w'(a,b) \\ &\geq 0. \end{aligned}$$

Somit haben wir nachgewiesen, dass $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$.

Wir weisen jetzt D1 nach. Da wegen S1 und S2 gilt

$$\min\{w'(a,a) \mid a \in \Sigma\} \geq 0 \geq \max\{w'(a,b) \mid a \neq b \in \overline{\Sigma}\},$$

folgt, dass

$$\begin{aligned}
\max\{w(a,a) \mid a \in \Sigma\} &= \max\{D - w'(a,a) \mid a \in \Sigma\} \\
&= D - \min\{w'(a,a) \mid a \in \Sigma\} \\
&\leq D - \max\{w'(a,b) \mid a \neq b \in \overline{\Sigma}\} \\
&= \min\{D - w'(a,b) \mid a \neq b \in \overline{\Sigma}\} \\
&= \min\{w(a,b) \mid a \neq b \in \overline{\Sigma}\}.
\end{aligned}$$

D2 folgt nach Definition von w unmittelbar aus S3.

Wir müssen nur noch die Dreiecksungleichung beweisen: $w(x,z) \leq w(x,y) + w(y,z)$ für alle $x,y,z \in \Sigma$. Dies geschieht durch eine vollständige Fallunterscheidung.

Fall 1 ($\boldsymbol{x = y = z}$): Dann gilt offensichtlich $w(x,x) \leq w(x,x) + w(x,x)$, da $w(x,x) \geq 0$.

Fall 2 ($\boldsymbol{x = y \neq z}$): Dann gilt

$$w(x,z) = w(y,z) \leq w(x,y) + w(y,z),$$

da wir wissen, dass $w(x,y) = w(x,x) \geq 0$.

Fall 3 ($\boldsymbol{x = z \neq y}$): Es gilt nun mit Hilfe von S3, dass

$$w(x,z) = w(x,x) = D - w'(x,x) \quad \text{und} \quad w(x,y) + w(y,z) = 2w(y,z) = 2D - w'(y,z).$$

Wir müssen also nur noch zeigen, dass

$$D - w'(x,x) \leq 2D - 2w'(y,z) \quad \text{bzw.} \quad 2w'(y,z) \leq D + w'(x,x)$$

gilt. Dies gilt aber offensichtlich, da $D \geq 0$ und da nach Definition eines Ähnlichkeitsmaßes $w'(y,z) \leq 0$ für $y \neq z \in \overline{\Sigma}$ und $w'(x,x) \geq 0$ für $x \in \Sigma$ gilt.

Fall 4 ($\boldsymbol{|\{x,y,z\}| = 3}$): Es gilt wiederum

$$w(x,z) = D - w'(x,z) \quad \text{und} \quad w(x,y) + w(y,z) = 2D - w'(x,y) - w'(y,z).$$

Damit genügt es zu zeigen, dass $w'(x,y) + w'(y,z) \leq D + w'(x,z)$ gilt. Nach Definition eines Ähnlichkeitsmaßes gilt $w'(x,y) + w'(y,z) \leq 0$ für paarweise verschiedene $x,y,z \in \overline{\Sigma}$. Nach Wahl von D gilt außerdem, dass $D + w'(x,z) \geq 0$ ist. Damit ist der Beweis abgeschlossen. ∎

Damit haben wir trotz der unterschiedlichen Definition gesehen, dass Distanzen und Ähnlichkeiten eng miteinander verwandt sind. Dies unterstreicht insbesondere auch noch einmal der folgende Satz.

Theorem 3.19 *Sei $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ eine sinnvolle Kostenfunktion für ein Distanzmaß d und sei $C \in \mathbb{R}_+$ so gewählt, dass $C - w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}$ eine sinnvolle Kostenfunktion für ein Ähnlichkeitsmaß s ist und dass gilt:*

$$\forall x, y \in \Sigma : \quad C \leq w(x, -) + w(y, -) - w(x, y). \tag{3.1}$$

Dann gilt für ein Alignment $(\overline{a}, \overline{b})$ für $a, b \in \Sigma^*$:

$$\overline{d}(a, b) + s(a, b) = C \cdot \ell$$

für eine geeignet gewählte Konstante $\ell \in \mathbb{N}_0$.

Beweis: Für ein beliebiges Alignment $(\overline{a}, \overline{b})$ für $a, b \in \Sigma^*$ gilt offensichtlich, dass

$$w(\overline{a}, \overline{b}) + s(\overline{a}, \overline{b}) = w(\overline{a}, \overline{b}) + (C - w)(\overline{a}, \overline{b}) = w(\overline{a}, \overline{b}) + C \cdot |\overline{a}| - w(\overline{a}, \overline{b}) = C \cdot |\overline{a}|.$$

Beachte hierbei, dass $|\overline{a}| = |\overline{b}|$. Da nach Definition

$$\begin{aligned} \overline{d}(a, b) &= \min \left\{ w(\overline{a}, \overline{b}) \ : \ (\overline{a}, \overline{b}) \text{ ist Alignment für } a, b \right\}, \\ s(a, b) &= \max \left\{ (C - w)(\overline{a}, \overline{b}) \ : \ (\overline{a}, \overline{b}) \text{ ist Alignment für } a, b \right\}, \end{aligned}$$

gilt, müssen wir nur noch zeigen, dass das Minimum beim Distanzmaß und das Maximum beim Ähnlichkeitsmaß an denselben Stellen angenommen wird.

Wir nehmen für einen Widerspruchsbeweis an, es gäbe bezüglich des Ähnlichkeitsmaßes ein besseres Alignment. Sei $(\overline{a}, \overline{b})$ ein optimales Alignment für a und b für das Distanzmaß, d.h. $w(\overline{a}, \overline{b}) = \overline{d}(a, b)$. Dann gilt $s(a, b) > (C - w)(\overline{a}, \overline{b})$ und es gibt ein optimales Alignment $(\tilde{a}, \tilde{b})$ für a und b mit $s(a, b) = (C - w)(\tilde{a}, \tilde{b})$. Also gilt $(C - w)(\overline{a}, \overline{b}) < (C - w)(\tilde{a}, \tilde{b})$ und somit (mit $|\overline{a}| = |\overline{b}|$ und $|\tilde{a}| = |\tilde{b}|$)

$$w(\tilde{a}, \tilde{b}) < w(\overline{a}, \overline{b}) + C(|\tilde{a}| - |\overline{a}|). \tag{3.2}$$

Da nach Voraussetzung $w(\overline{a}, \overline{b}) \leq w(\tilde{a}, \tilde{b})$ gilt, folgt mit 3.2 sofort $|\overline{a}| < |\tilde{a}|$. Somit hat das Alignment $(\tilde{a}, \overline{a})$ mehr Indel-Stellen als das Alignment $(\overline{a}, \overline{b})$. Somit müssen Matches oder Substitutionen durch Insertionen und Deletionen ersetzt worden sein. Die Ungleichung 3.1 impliziert jedoch, dass jede solche Verlängerung eines Alignments die Distanz um mindestens C erhöht. Somit gilt

$$w(\tilde{a}, \tilde{b}) \geq w(\overline{a}, \overline{b}) + C(|\tilde{a}| - |\overline{a}|) \tag{3.3}$$

Aus den Ungleichungen 3.2 und 3.3 folgt sofort

$$w(\overline{a}, \overline{b}) + C(|\tilde{a}| - |\overline{a}|) > w(\tilde{a}, \tilde{b}) \geq w(\overline{a}, \overline{b}) + C(|\tilde{a}| - |\overline{a}|),$$

was offensichtlich ein Widerspruch ist. Die umgekehrte Richtung lässt sich analog beweisen. ■

Wir merken hier noch an, dass die Ungleichung 3.1 im Satz äquivalent zu der Bedingung $s(x, y) \geq s(x, -) + s(-, y)$ für alle $x, y \in \Sigma$ ist. Somit wird hier nur gefordert, dass auch im Ähnlichkeitsmaß eine Substitution besser bewertet wird als deren Ersetzung durch eine Insertion und eine Deletion.

Unter gewissen Voraussetzungen kann man also beliebig zwischen Edit-, Alignment-Distanz oder Ähnlichkeitswerten wechseln, ohne in der Bewertung der Ähnlichkeit von Sequenzen andere Ergebnisse zu erhalten.

3.2 Bestimmung optimaler globaler Alignments

In diesem Abschnitt geht es nun darum, optimale Alignments für zwei Zeichenreihen tatsächlich zu berechnen. Dabei werden die beiden Zeichenreihen zueinander „aligned", also so zueinander ausgerichtet, dass sie danach dieselbe Länge haben und so ähnlich wie möglich sind (bezüglich eines geeignet gewählten Distanz- oder Ähnlichkeitsmaßes).

Geg.: $s = s_1 \cdots s_n \in \Sigma^n$, $t = t_1 \cdots t_m \in \Sigma^m$ und ein Distanz- oder Ähnlichkeitsmaß d.
Ges.: Ein optimales Alignment $(\overline{s}, \overline{t})$ für s, t.

Wir werden uns zunächst hauptsächlich mit Distanzmaßen beschäftigen. Es ist meist offensichtlich, wie die Methoden für Ähnlichkeitsmaße zu modifizieren sind. Wir fordern den Leser an dieser Stelle ausdrücklich auf, dies auch jedes Mal zu tun.

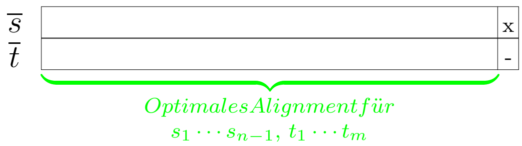
3.2.1 Der Algorithmus nach Needleman-Wunsch

Wir nehmen an, wir kennen schon ein optimales Alignment $(\overline{s}, \overline{t})$ für s und t. Es gibt jetzt drei Möglichkeiten, wie die letzte Spalte $(\overline{t}_{|\overline{t}|}, \overline{s}_{|\overline{s}|})$ dieses optimalen Alignments aussehen kann:

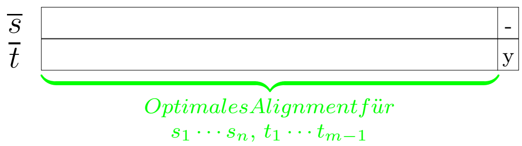
1. $x = s_n$ wurde durch $y = t_m$ substituiert:

$$\begin{array}{c} \overline{s} \\ \overline{t} \end{array} \underbrace{\begin{array}{|c|}\hline \qquad\qquad\qquad\qquad \\ \hline \qquad\qquad\qquad\qquad \\ \hline \end{array}}_{\substack{Optimales Alignment für \\ s_1 \cdots s_{n-1},\ t_1 \cdots t_{m-1}}} \!\!\begin{array}{|c|}\hline x \\ \hline y \\ \hline \end{array}$$

2. Das letzte Zeichen $x = s_n$ in s wurde gelöscht:



3. Das letzte Zeichen $y = t_m$ in t wurde eingefügt:

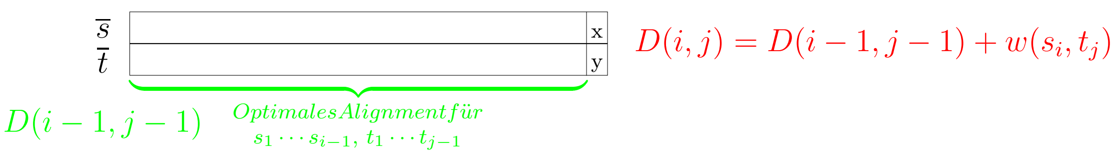


In allen drei Fällen überlegt man sich leicht, dass das Alignment, das durch Streichen der letzten Spalte entsteht, also $(\overline{s}_1 \cdots \overline{s}_{|\overline{s}|-1}, \overline{t}_1 \cdots \overline{t}_{|\overline{t}|-1})$, ebenfalls ein optimales Alignment für $s_1 \cdots s_{n-1}$ mit $t_1 \cdots t_{m-1}$, $s_1 \cdots s_{n-1}$ mit $t_1 \cdots t_m$ bzw. $s_1 \cdots s_n$ mit $t_1 \cdots t_{m-1}$ sein muss. Gäbe es andernfalls ein besseres Alignment (mit geringerer Distanz), so könnte man daraus mit der Edit-Operation an der letzten Stelle ein besseres Alignment für s mit t konstruieren.

Mit diesen Vorüberlegungen können wir das Verfahren von Needleman-Wunsch formulieren, das ein optimales Alignment für zwei Sequenzen $s = s_1 \cdots s_n \in \Sigma^n$ und $t = t_1 \cdots t_m \in \Sigma^m$ berechnet. Dazu wird eine Matrix $D(i, j)$ aufgestellt, in welcher jeweils die Distanz eines optimalen Alignments für $s_1, \ldots s_i$ und $t_1, \ldots t_j$ abgespeichert wird. Die Matrix kann rekursiv mit der folgenden Rekursionsformel berechnet werden:

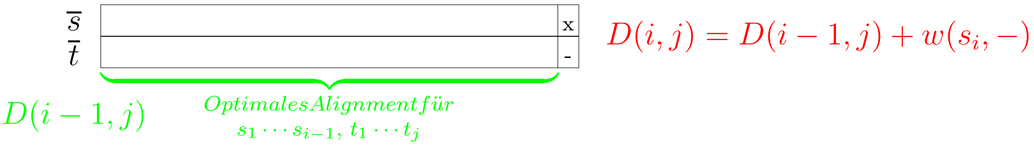
$$D(i, j) \quad = \quad \min \left\{ \begin{array}{lcl} D(i-1, j-1) & + & w(s_i, t_j), \\ D(i-1, j) & + & w(s_i, -), \\ D(i, j-1) & + & w(-, t_j) \end{array} \right\}.$$

Die folgenden Bilder illustrieren nochmals obige Rekursionsformel. Im ersten Fall ist das optimale Alignment für $s_1 \cdots s_{i-1}$ und $t_1 \cdots t_{j-1}$ bereits berechnet und in $D(i-1, j-1)$ abgelegt. Um nun die Distanz eines Alignments für $s_1 \cdots s_i$ und $t_1 \cdots t_j$ zu erhalten, müssen noch die Kosten für die Substitution von s_n durch t_m hinzuaddieren:

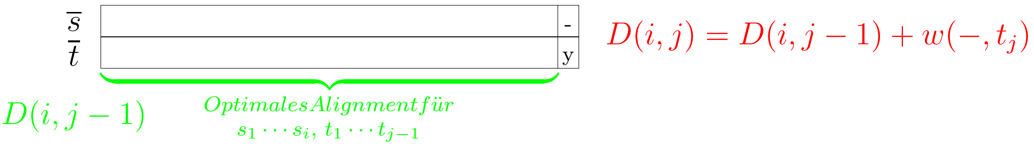


Im zweiten Fall wurde ein Zeichen in t gelöscht. Um nun die Distanz eines Alignments für $s_1 \cdots s_i$ und $t_1 \cdots t_j$ zu erhalten, muss zu den Kosten dieser Löschung noch die

Distanz des bereits berechneten optimalen Alignments für $s_1 \cdots s_{i-1}$ und $t_1 \cdots t_j$ dazuaddiert werden.



Im letzten Fall wurde ein Zeichen in die Sequenz t eingefügt. Wie bei den anderen beiden Fällen auch, müssen zur Distanz des bereits berechneten optimalen Alignments für $s_1 \cdots s_i$ und $t_1 \cdots t_{j-1}$, noch die Kosten für die Einfügung hinzuaddiert werden, um die Distanz eines Alignments für $s_1 \cdots s_i$ und $t_1 \cdots t_j$ zu erhalten.



Da das Optimum, wie vorher schon erläutert, einer dieser Fälle ist, genügt es, aus allen drei möglichen Werten das Minimum auszuwählen. Im Falle von Ähnlichkeitsmaßen wird dann entsprechend das Maximum genommen.

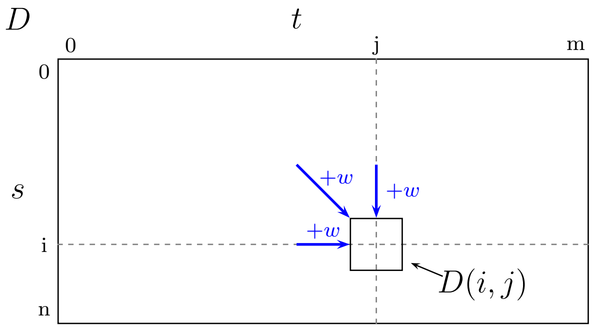


Abbildung 3.6: Skizze: Berechnung optimaler Alignments nach Needleman-Wunsch

Abbildung 3.6 zeigt schematisch, wie der Wert eines Eintrags $D(i,j)$ in der Matrix D von den anderen Werten aus D abhängt. Nun fehlen nur noch die zugehörigen Anfangswerte:

$$D(0,0) = 0, \qquad D(i,0) = \sum_{k=1}^{i} w(s_k,-), \qquad D(0,j) = \sum_{k=1}^{j} w(-,t_k).$$

Die Korrektheit folgt aus der Überlegung, dass in der ersten Spalte $D(i,0)$ die Abstände optimaler Alignments von $s_1 \cdots s_i$ mit $t_1 \cdots t_0 = \varepsilon$ stehen. Es bleibt einem

$\underline{\text{SEQUENCEALIGNMENT (char } s[], \text{ int } n, \text{ char } t[], \text{ int } m)}$

$\{$

$\quad D[0,0] = 0;$

$\quad \textbf{for } (i = 1;\ i \leq n;\ i{+}{+})$

$\qquad D[i,0] = D[i-1,0] + w(s_i, -);$

$\quad \textbf{for } (j = 1;\ j \leq m;\ j{+}{+})$

$\qquad D[0,j] = D[0,j-1] + w(-, t_j);$

$\quad \textbf{for } (i = 1;\ i \leq n;\ i{+}{+})$

$\qquad \textbf{for } (j = 1;\ j \leq m;\ j{+}{+})$

$$D[i,j] = \min \left\{ \begin{array}{l} D[i-1,j] + w(s_i, -), \\ D[i,j-1] + w(-, t_j), \\ D[i-1,j-1] + w(s_i, t_j) \end{array} \right\};$$

$\}$

Abbildung 3.7: Algorithmus: Verfahren von Needleman und Wunsch

gar nichts anderes übrig, als alle Zeichen aus $s_1 \cdots s_j$ zu löschen. Analoges gilt für die erste Zeile.

Folgendes Beispiel zeigt ausführlich für zwei Sequenzen s und t, wie deren optimale Alignment-Distanz bestimmt wird.

$$\begin{array}{ccccccccc} s & = & A & G & G & C & T & G & \\ t & = & A & C & C & G & G & T & A \end{array}$$

In den nachfolgenden Abbildungen gilt, dass die Zeichenreihe s immer vertikal und die Zeichenreihe t immer horizontal aufgetragen ist.

Der erste Schritt besteht darin, den so genannten „*Edit-Graphen*“ aufzustellen. Dazu wird die Sequenz t horizontal und s vertikal aufgetragen. Anschließend werden in den Graphen Pfeile eingefügt, abhängig davon, ob an der jeweiligen Stelle eine Substitution, eine Löschung, eine Einfügung oder aber ein Match auftritt. Ein horizontaler bzw. vertikaler blauer Pfeil steht für eine Insertion bzw. für eine Deletion, ein roter Pfeil für eine Substitution und schließlich ein grüner Pfeil für einen Match.

Daraufhin wird der Edit-Graph „mit Zahlen gefüllt“. An die jeweilige Stelle im Graphen wird die momentane Distanz des jeweiligen Alignments eingetragen, berechnet mit der Rekursionformel von weiter oben. Im Beispiel wird von einer Kostenfunktion ausgegangen, welche einer Löschung und einer Einfügung die Kosten 2 zuordnet, und bei welcher eine Substitution die Kosten 3 verursacht. Ein Match verursacht natürlich keine Kosten.

In der rechten unteren Ecke, also in $D(n,m)$, steht nun die Distanz eines optimalen Alignments für s und t. Damit haben wir zwar den Wert eines optimalen Alignments

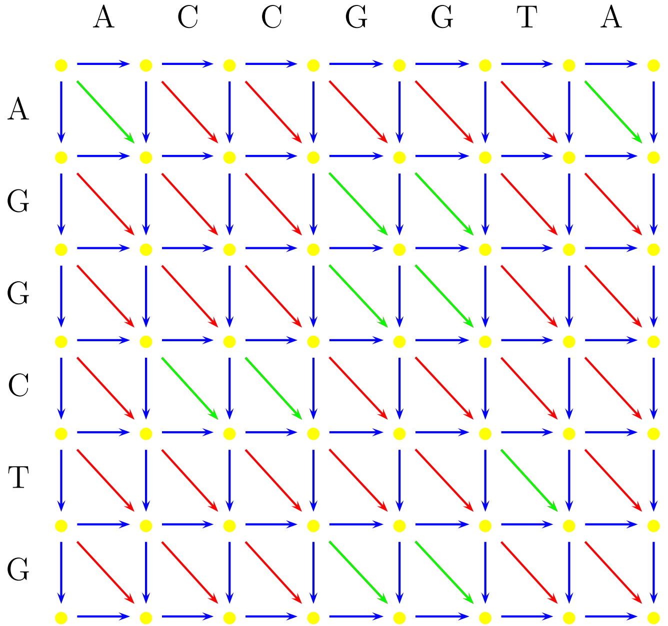
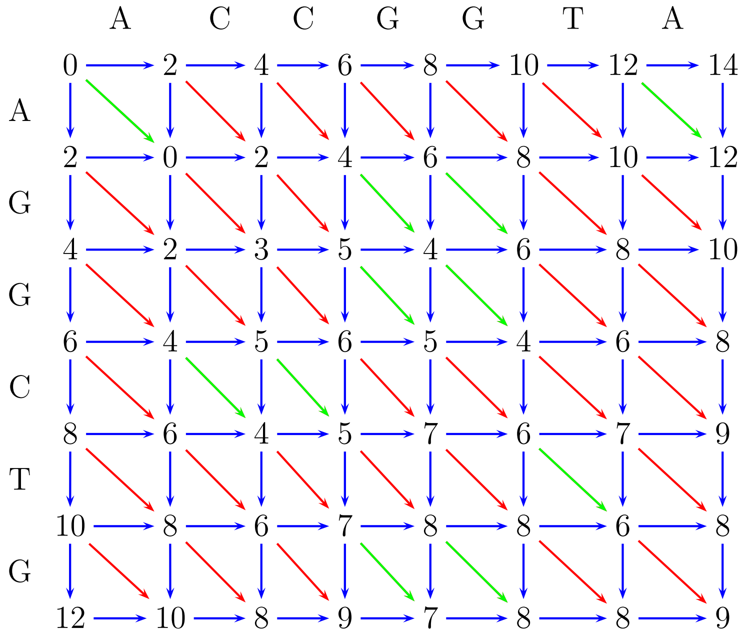


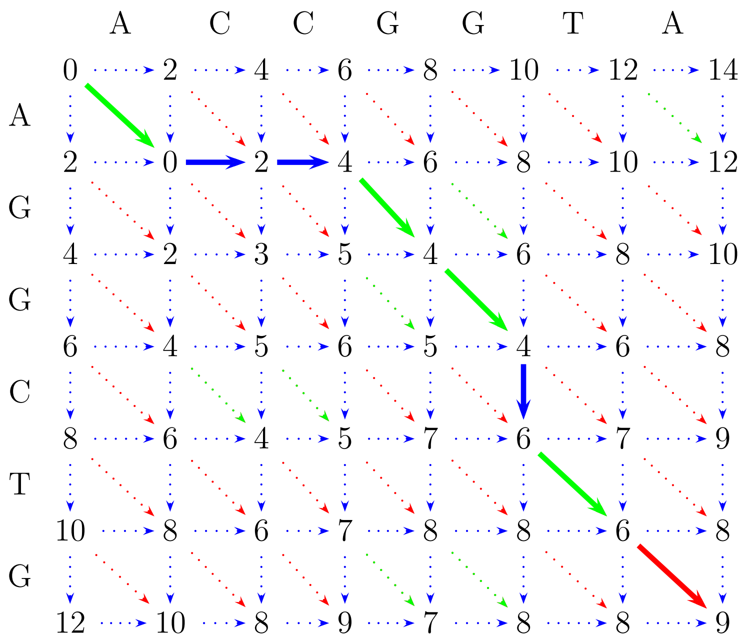
Abbildung 3.8: Skizze: Edit-Graph für s und t ohne Distanzen



Match = 0, Indel = 2, Subst = 3

Abbildung 3.9: Skizze: Edit-Graph für s und t mit Distanzen für s und t bestimmt, kennen das Alignment an sich jedoch noch nicht. Um nun dieses zu erhalten, wird ein Pfad im Graphen von rechts unten nach links oben gesucht, der minimale Kosten verursacht.

Dieser Pfad wird folgendermaßen gefunden. Gestartet wird in der rechten unteren Ecke. Als Vorgängerknoten wird nun der Knoten gewählt, der zuvor als Sieger bei der Minimum-Bildung hervorging. Liefern mehrere Knoten die gleichen minimalen Kosten, kann einer davon frei gewählt werden. Meist geht man hier in einer vorher fest vorgegeben Reihenfolge bei Unentschieden vor, z.B. Insertion vor Substitution vor Deletion. So verfährt man nun immer weiter, bis man in der linken oberen Ecke ankommt.



Match = 0, Indel = 2, Subst = 3

Abbildung 3.10: Skizze: Pfad im Edit-Graphen zur Bestimmung des Alignments

Nun ist es nicht mehr schwer, das optimale Alignment für s und t anzugeben. Dieses muss nur noch aus dem Edit-Graphen (entlang des gefundenen Pfades) abgelesen werden, wie dies in Abbildung 3.11 dargestellt ist.

$$\begin{array}{lcccccccc} s: & A & - & - & G & G & C & T & G \\ t: & A & C & C & G & G & - & T & A \end{array}$$

Abbildung 3.11: Beispiel: Optimales globales Alignment von s mit t

Fassen wir zum Abschluss dieses Abschnittes noch das Ergebnis zusammen.

Theorem 3.20 *Das optimale globale paarweise Sequenzen Alignment für s und t mit $n = |s|$ und $m = |t|$ sowie die zugehörige Alignment-Distanz lassen sich in Zeit $O(nm)$ und mit Platz $O(nm)$ berechnen.*

Zum Schluss dieses Abschnitts wollen wir noch anmerken, dass das algorithmische Lösen von Rekursionsgleichungen mit Hilfe von Tabellen *dynamische Programmierung* genannt wird. Normalerweise würde man Rekursionsgleichungen mit Hilfe von Rekursionen lösen. Werden hierbei jedoch sehr oft gleiche Teilprobleme rekursiv gelöst, so ist dies sehr ineffizient. Mit Hilfe der Tabellen werden dabei die Ergebnisse von bereits gelösten Teilproblemen gespeichert, so dass sie wiederverwendet werden können, was in diesen Fällen die Effizienz erheblich erhöht.

Musterbeispiel hierfür ist die Berechnung der n-ten Fibonacci-Zahl f_n die rekursiv durch $f_n = f_{n-1} + f_{n-2}$ und $f_1 = 1$ und $f_0 = 0$ definiert ist. Würde man diese Rekursionsgleichung rekursiv lösen, so würde man exponentiell oft den Wert f_2 berechnen (das gilt im Prinzip für alle Werte). Somit würde eine rekursive Lösung exponentiellen Aufwand in n benötigen. Berechnet man hingegen die Fibonacci-Zahlen mit Hilfe der dynamischen Programmierung, so füllt man eine Tabelle $F(i)$ für $i = 2, \ldots, n$ aus. Dafür sind nur linear (in n) viele Additionen nötig.

3.2.2 Sequenzen Alignment mit linearem Platz (Modifikation von Hirschberg)

Bisher wurde zur Bestimmung eines optimalen Alignments für s und t Platz in der Größenordnung $O(nm)$ benötigt. Dies soll nun dahingehend optimiert werden, dass nur noch linear viel Platz gebraucht wird.

Es fällt auf, dass während der Berechnung der $D(i, j)$ immer nur die momentane Zeile i und die unmittelbar darüber liegende Zeile $i - 1$ benötigt wird. Somit bietet es sich an, immer nur diese beiden relevanten Zeilen zu speichern und somit nur linear viel Platz zu beanspruchen. Der Algorithmus in Abbildung 3.12 beschreibt genau dieses Verfahren.

Mit dem oben beschriebenen Verfahren lässt sich die Distanz der beiden Sequenzen s und t mit linearem Platz berechnen. Allerdings hat das Verfahren den Haken, dass das Alignment selbst nicht mehr einfach anhand des Edit-Graphen aufgebaut werden kann, da ja die nötigen Zwischenergebnisse nicht gespeichert wurden.

SequenceAlignment (char $s[]$, int n, char $t[]$, int m)

$\{$
 $D[0] = 0;$
 for $(j = 0;\ j \leq m;\ j{+}{+})$
  $D[j] = D[j-1] + w(-, t_j);$
 for $(i = 1;\ i \leq n;\ i{+}{+})$
 $\{$
  for $(j = 0;\ j \leq m;\ j{+}{+})$
   $D'[j] = D[j];$
  $D[0] = D'[0] + w(s_i, 0);$
  for $(j = 1;\ j \leq m;\ j{+}{+})$

$$D[j] = \min \left\{ \begin{array}{l} D'[j] + w(s_i, -), \\ D[j-1] + w(-, t_j), \\ D'[j-1] + w(s_i, t_j) \end{array} \right\};$$

 $\}$
$\}$

Abbildung 3.12: Algorithmus: platzsparende Variante von Needleman-Wunsch

Mit dem Verfahren nach Hirschberg kann ein optimales Sequenzen Alignment selbst konstruiert werden, so dass nur linear viel Platz benutzt werden muss. Dazu betrachten wir zunächst einmal ein optimales paarweises Alignment von s mit t, wie in der folgenden Abbildung 3.13 angegeben. Wir teilen nun dieses Alignment so in zwei

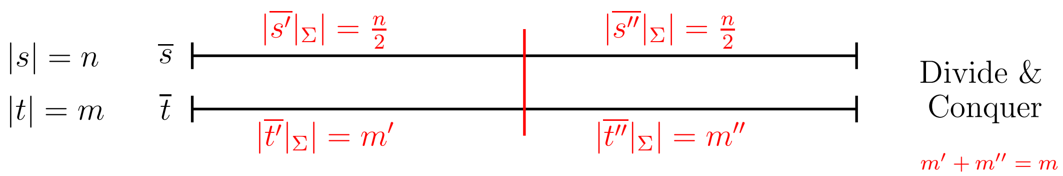


Abbildung 3.13: Skizze: Optimales Alignment für s und t

Teil-Alignments auf, dass beide Teile in etwa die Hälfte der Zeichen aus s enthalten: der erste Teil enthalte $\lceil n/2 \rceil$ und der zweite Teil $\lfloor n/2 \rfloor$ Zeichen aus s. Um uns im Folgenden das Leben etwas leichter zu machen, nehmen wir an, dass n gerade ist und wir somit ohne die Gauß-Klammern weiter arbeiten können.

Wir merken an dieser Stelle noch an, dass dieser Aufteilungsschritt nicht eindeutig sein muss, da das Alignment $\overline{s}$ von s sehr viele Leerzeichen zwischen dem Zeichen $s_{n/2}$ und dem Zeichen $s_{n/2+1}$ enthalten kann.

Im Folgenden bezeichnen wir mit m' die Anzahl der Zeichen aus t die bei der Aufteilung in der ersten Hälfte des Alignments sind und mit m'' die Anzahl der Zeichen in der zweiten Hälfte. Es gilt also $m = m' + m''$. Weiter bezeichne $\overline{s'}$ und $\overline{s''}$ bzw. $\overline{t'}$ und $\overline{t''}$ die Teile des Alignments nach der Aufteilung, d.h. $\overline{s} = \overline{s'} \cdot \overline{s''}$ und $\overline{t} = \overline{t'} \cdot \overline{t''}$. Ferner gilt $s' = \overline{s'}|_\Sigma$ und $s'' = \overline{s''}|_\Sigma$ bzw. $t' = \overline{t'}|_\Sigma$und $t'' = \overline{t''}|_\Sigma$. Es gilt also $s = s' \cdot s''$ und $t = t' \cdot t''$ sowie $|s'| = |s''| = n/2$ und $|t'| = m'$ bzw. $|t''| = m''$.

Zuerst einmal bemerken wir, dass sowohl $(\overline{s'}, \overline{t'})$ ein optimales Alignment für s' mit t' sein muss, als dass auch $(\overline{s''}, \overline{t''})$ ein optimales Alignment für s'' mit t'' sein muss. Auch hier könnten wir andererseits aus besseren Alignments für s' mit t' bzw. s'' mit t'' ein besseres Alignment für s mit t konstruieren.

Dies führt uns unmittelbar auf die folgende *algorithmische Idee:* Berechne optimale Alignments für $s_1 \cdots s_{n/2}$ mit $t_1 \cdots t_{m'}$ sowie für $s_{n/2} \cdots s_n$ mit $t_{m'+1} \cdots t_m$. Dieser Ansatz führt uns auf einen Divide-and-Conquer-Algorithmus, da wir nun rekursiv für kleinere Eingaben ein Problem derselben Art lösen müssen.

Der *Conquer-Schritt* ist dabei trivial, da wir einfach die beiden erhalten Alignments für beide Teile zu einem großen Alignment für s mit t zusammenhängen müssen, wie dies in der folgenden Abbildung 3.14 dargestellt ist.

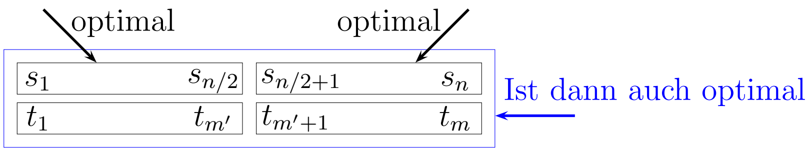


Abbildung 3.14: Skizze: Conquer-Schritt des Hirschberg-Algorithmus

Allerdings haben wir nun noch ein kleines Problem übersehen. Wir kennen nämlich m' noch nicht. Wir haben m' ja über ein optimales Alignment für s mit t definiert. Wenn wir also erst das optimale Alignment für s mit t berechnen wollen, kennen wir $m' = |t'|$ ja noch nicht.

Wie finden wir m' jetzt also? Ein erster naiver Gedanke ist, dass man alle möglichen $m' \in [0 : m]$ ausprobiert. Dies sind allerdings recht viele, die uns ja dann auch noch $m + 1$ rekursiv zu lösende Teilprobleme zur Aufgabe stellen.

So dumm, wie der naive Ansatz jedoch zuerst klingt, ist er gar nicht, denn wir wollen ja gar nicht die Alignments selbst berechnen, sondern nur wissen, wie m' für ein optimales Alignment von s mit t aussieht. Für die weitere Argumentation werden wir die folgende Abbildung 3.15 zu Hilfe nehmen. In dieser Abbildung ist die Tabelle $D(i, j)$ wieder als Edit-Graph bildlich dargestellt. Ein optimales Alignment entspricht in diesem Graphen einem Weg von $(0, 0)$ nach (n, m). Offensichtlich muss dieser Weg die Zeile $n/2$ an einem Punkt m' schneiden. Wir merken hier nochmals an, dass dieser Punkt nicht eindeutig sein muss, da aufgrund von Insertionen der Pfad waagerecht innerhalb der Zeile $n/2$ verlaufen kann.

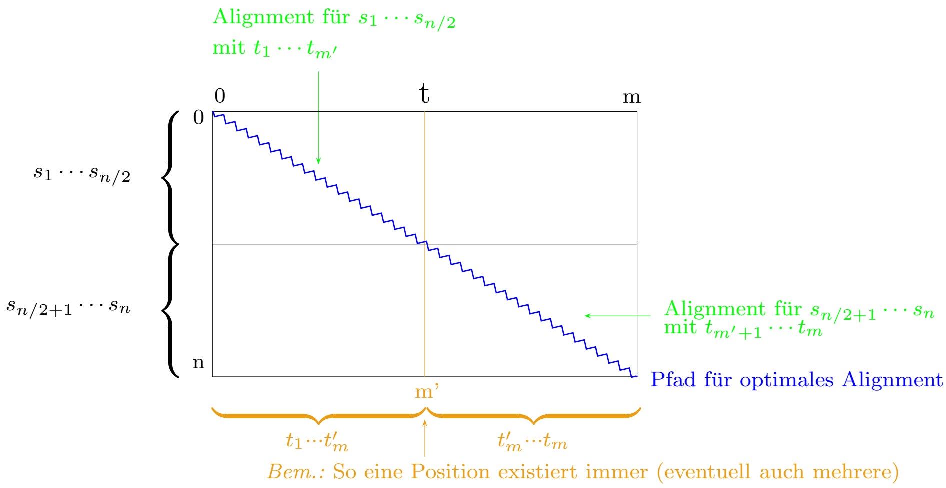


Abbildung 3.15: Skizze: Auffinden von m' (Divide-Schritt bei Hirschberg)

Der Pfad von $(0, 0)$ nach (n, m) zerfällt also in zwei Teile, nämlich in $(0, 0)$ bis $(n/2, m')$ und in $(n/2, m')$ nach (n, m). Diese beiden Teile entsprechen dann genau den vorher diskutierten optimalen Alignments für s' mit t' und s'' mit t''.

Können wir die beiden Teilpfade jetzt schnell finden? Den ersten auf jeden Fall. Wir berechnen die optimalen Alignment-Distanzen von $s_1 \cdots s_{n/2}$ mit $t_1 \cdots t_{m'}$ für alle $m' \in [0 : m]$. Dies können wir mit unserem vorhin in Abbildung 3.12 vorgestellten Algorithmus in linearem Platz berechnen. Dort haben wir als Endergebnis das Feld $D[j]$ erhalten, dass die Alignment-Distanzen von s zu allen Präfixen von t enthielt.

Jetzt brauchen wir noch den zweiten Teil des Pfades. Dazu benötigen wir insbesondere die Alignment-Distanzen von $s_{n/2} \cdots s_n$ mit $t_{m'} \cdots t_m$ für alle $m' \in [0 : m]$. Diese können wir jedoch mit demselben Algorithmus berechnen. Wir stellen uns die Tabelle nur um 180 Grad um den Mittelpunkt gedreht vor. Wir berechnen dann alle Alignment-Distanzen von $(s'')^R = s_n \cdots s_{n/2}$ mit $(t'')^R = t_m \cdots t_{m'}$, wobei hier x^R für eine Zeichenreihe $x = x_1 \cdots x_n$ die *gespiegelte oder reversierte Zeichenreihe* $x^R = x_n \cdots x_1$ bezeichnet.

Damit die Korrektheit dieses Ansatzes gilt, müssen wir nur den folgenden Satz beweisen.

Theorem 3.21 *Sei* $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ *eine Kostenfunktion und seien* $s, t \in \Sigma^*$, *dann gilt* $\overline{d}_w(s, t) = \overline{d}_w(s^R, t^R)$.

Beweis: Es gilt für beliebige $\overline{s}, \overline{t} \in \overline{\Sigma}^*$ mit $\overline{s}|_\Sigma = s$ und $\overline{t}|_\Sigma = t$:

$$\begin{aligned} w(\overline{s}, \overline{t}) &= \sum_{i=1}^{|\overline{s}|} w(\overline{s}_i, \overline{t}_i) \\ &= \sum_{i=1}^{|\overline{s}|} w(\overline{s}_{|\overline{s}|-i+1}, \overline{t}_{|\overline{t}|-i+1}) \\ &= w(\overline{s^R}, \overline{t^R}). \end{aligned}$$

Somit gilt auch:

$$\begin{aligned} \overline{d}_w(s,t) &= \min\{w(\overline{s}, \overline{t}) \mid (\overline{s}, \overline{t}) \text{ ist ein Alignment für } s, t\} \\ &= \min\{w(\overline{s^R}, \overline{t^R}) \mid (\overline{s^R}, \overline{t^R}) \text{ ist ein Alignment für } s^R, t^R\} \\ &= \overline{d}_w(s^R, t^R). \end{aligned}$$

■

Bezeichne $V(n/2, k)$ die minimale Alignment-Distanz von $s' = s_1 \cdots s_{n/2}$ mit $t_1 \cdots t_k$ und $V'(n/2, k)$ die minimale Alignment-Distanz von $s'' = s_{n/2+1} \cdots s_n$ mit $t_k \cdots t_m$ was nach obigem Satz gleichbedeutend mit der minimalen Alignment-Distanz von $(s'')^R$ mit $t_m \cdots t_k$ ist.

$$\begin{aligned} V(\frac{n}{2}, k) &= \overline{d}(s_1 \cdots s_{n/2}, t_1 \cdots t_k) \\ V'(\frac{n}{2}, k) &= \overline{d}(s_{n/2+1} \cdots s_n, t_{k+1} \cdots t_m) = \overline{d}(s_n \cdots s_{n/2+1}, t_m \cdots t_{k+1}) \end{aligned}$$

Nach unseren Überlegungen gilt für das optimale m', dass für die optimale Edit-Distanz gilt: $\overline{d}(s,t) = V(n/2, m') + V'(n/2, m')$. Wir können also $\overline{d}(s,t)$ und m' wie folgt berechnen:

$$\begin{aligned} \overline{d}(s,t) &= \min\left\{V(\frac{n}{2}, k) + V'(\frac{n}{2}, k)) \;:\; k \in [0:m]\right\} \\ m' &= \operatorname{argmin}\left\{V(\frac{n}{2}, k) + V'(\frac{n}{2}, k)) \;:\; k \in [0:m]\right\} \end{aligned}$$

Hierbei bezeichnet argmin einen Index-Wert, für den in der Menge das Minimum angenommen wird, d.h. es gilt für eine Menge $M = \{e_i \;:\; i \in I\}$ mit der zugehörigen Indexmenge I:

$$\min\{e_i \;:\; i \in I\} = e_{\operatorname{argmin}\{e_i : i \in I\}}.$$

Somit können wir also für zwei Zeichenreihen s und t den Schnittpunkt m' berechnen, der zwei optimale Teil-Alignments angibt, aus dem ein optimales Alignment für s und t berechnet wird.

In der folgenden Abbildung 3.16 ist der vollständige Hirschberg-Algorithmus angegeben. Wir bestimmen also zunächst den „Mittelpunkt des Pfades eines optimalen Alignments“ $(n/2, m')$, dann lösen wir rekursiv die beiden entstehenden Alignment-Probleme und konstruieren zum Schluss aus diesen beiden Alignments eine neues optimales Alignment für s und t.

1. Berechne die Werte optimaler Alignments für $s' = s_1 \cdots s_{n/2}$ mit $t_1 \cdots t_k$ für alle $k \in [0 : m]$, d.h. $V(n/2, k)$ für alle $k \in [0 : m]$.

   (In Wirklichkeit $s_1 \cdots s_{n/2}$ mit $t_1 \cdots t_m$.)

2. Berechne die Werte optimaler Alignments für $s'' = s_{n/2+1} \cdots s_n$ mit $t_k \cdots t_m$ für alle $k \in [0 : m]$, d.h. $V'(n/2, k)$ für alle $k \in [0 : m]$.

   (In Wirklichkeit $s_n \cdots s_{n/2+1}$ mit $t_m \cdots t_1$.)

3. Bestimme m' mittels $m' = \operatorname{argmin}\{V(\frac{n}{2}, k) + V'(\frac{n}{2}, k)) \mid k \in [0 : m]\}$.

4. Löse rekursiv die beiden Alignment-Probleme für $s' = s_1 \cdots s_{n/2}$ mit $t_1 \cdots t_{m'}$ sowie $s'' = s_{n/2} \cdots s_n$ mit $t_{m'+1} \cdots t_m$.

Abbildung 3.16: Algorithmus: Verfahren von Hirschberg

Wir müssen uns nur noch überlegen, wann wir die Rekursion abbrechen und ob sich diese Teilprobleme dann trivial lösen lassen. Wir brechen die Rekursion ab, wenn die erste Zeichenreihe, d.h. das Teilwort von s, die Länge 1 erreicht.

$$
\begin{array}{llcccl}
(s_l, t_p \cdots t_{p'}): & \text{für} & t_p \cdots t_{p'} & = & \varepsilon & \Rightarrow \binom{s_l}{-} \\
 & & t_p \cdots t_{p'} & \neq & \varepsilon & \Rightarrow \begin{pmatrix} -\cdots-s_l-\cdots- \\ t_p \cdots\cdots t_{p'} \end{pmatrix}
\end{array}
$$

$\hookrightarrow$ hier steht das Zeichen s_l sofern s_l in $t_p \cdots t_{p'}$ vorkommt, ansonsten ist die Position egal.

Folgendes Beispiel verdeutlicht die Vorgehensweise beim Verfahren von Hirschberg zur Bestimmung eines optimalen Sequenzen Alignments anhand von zwei Sequenzen s und t.

$$
\begin{array}{lcccccc}
s & = & A & G & G & T & \\
t & = & A & C & C & G & T
\end{array}
$$

Zuerst wird, wie oben beschrieben, der Wert des optimalen Alignments für $s_1 \cdots s_2$ mit $t_1 \cdots t_k$ und für $s_3 \cdots s_4$ mit $t_k \cdots t_5$ für alle $k \in [0 : 5]$ berechnet. Dies ist in Abbildung 3.17 bildlich dargestellt.

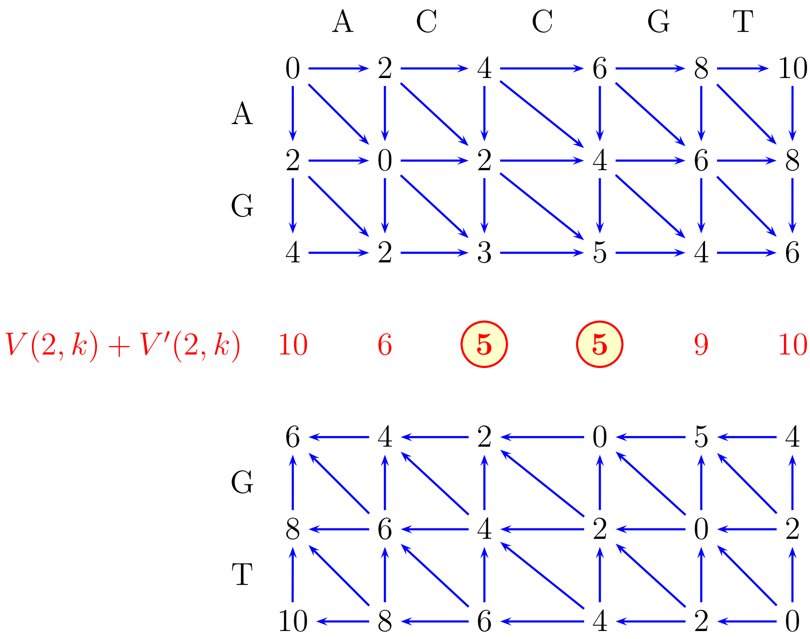


Abbildung 3.17: Beispiel: Bestimmung von m' im Hirschberg-Algorithmus

Der nächste Schritt besteht nun darin, m' zu bestimmen. In unserem Fall sind zwei verschiedene Werte möglich, da zweimal der Wert 5 auftritt. Für den weiteren Verlauf entscheiden wir uns für $m' = 3$. Jetzt müssen wir rekursiv die beiden Teile bearbeiten.

Zuerst betrachten wir den oberen linken Teil (siehe dazu auch Abbildung 3.18). Wieder haben wir zwei Schnittpunkte zur Wahl, nämlich 1 und 2. Wir entscheiden

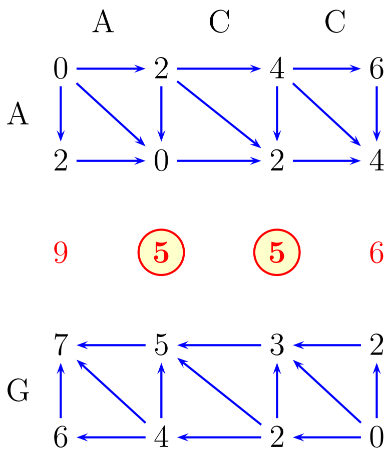


Abbildung 3.18: Beispiel: Erster rekursiver Aufruf im Hirschberg-Algorithmus

und für 1. Damit erhalten wir jetzt Probleme, bei denen die erste Sequenz Länge 1 hat. Wir müssen jetzt also ein Alignment für A mit A und für G mit CC finden. Offensichtlich wählt man $\binom{A}{A}$ und $\binom{G-}{CC}$. Dieses wird dann zu $\binom{AG-}{ACC}$ zusammengesetzt.

Jetzt fehlt noch der zweite rekursive Aufruf für $m' = 3$, d.h. der untere rechte Teil (siehe dazu Abbildung 3.19). Hier ist der Aufteilungspunkt eindeutig und die Zeichen

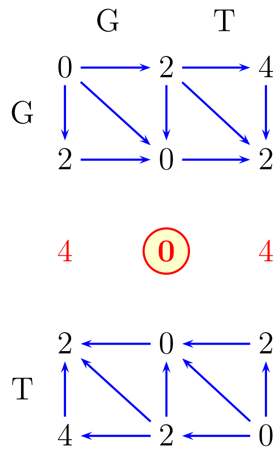


Abbildung 3.19: Beispiel: Zweiter rekursiver Aufruf im Hirschberg-Algorithmus

stimmen ja auch überein, so dass wir zuerst zwei kurze Alignments $\binom{G}{G}$ und $\binom{T}{T}$ erhalten, die dann zu $\binom{GT}{GT}$ zusammengesetzt werden.

Setzt man nun die beiden Alignments aus dem ersten Aufruf, nämlich $\binom{AG-}{ACC}$, und dem zweiten rekursiven Aufruf, nämlich $\binom{GT}{GT}$, zusammen, so erhalten wir als gesamtes Alignment $\binom{AG-GT}{ACCGT}$, das auch die schon berechnete Distanz 5 besitzt.

Wir haben bereits die Korrektheit der Variante von Hirschberg bewiesen. Es bleibt noch zu zeigen, dass der Platzbedarf wirklich linear ist und wie groß die Laufzeit ist.

Zuerst zum Platzbedarf: Dazu betrachten wir noch einmal den in Abbildung 3.16 angegebenen Algorithmus. Schritte 1 und 2 können wir, wie bereits erläutert, in linearem Platz $O(m)$ berechnen. Schritt 3 benötigt keinen weiteren Platz. Im letzten Schritt rufen wir zweimal rekursiv die Prozeduren auf und benötigen für die erste Rekursion Platz $O(m')$ sowie für die zweite Rekursion Platz $O(m'')$ und somit wieder Platz von $O(m' + m'') = O(m)$. Da wir den Platz aus Schritt 1 und 2 wiederverwenden können, benötigen wir insgesamt nur Platz $O(m)$.

Es bleibt die Laufzeitanalyse: Wir bezeichnen hierzu mit $T(n, m)$ den Zeitbedarf für die Hirschberg-Variante für zwei Zeichenreihen mit Längen n und m. Wir stellen zuerst eine Rekursionsformel für T auf.

Schritt 1 benötigt Laufzeit $c \cdot \frac{n}{2} \cdot m$ für eine geeignet gewählte Konstante c. Schritt 2 benötigt ebenfalls Laufzeit $c \cdot \frac{n}{2} \cdot m$. Schritt 3 benötigt $O(m)$ Operationen. Wir können hier ebenfalls, davon ausgehen, dass dies maximal $c \cdot m$ Operationen sind (im Zweifelsfall erhöhen wir c auf das Maximum der beiden Konstanten). Aufgrund der beiden rekursiven Aufrufe im Schritt 4, ist der Zeitbedarf hierfür durch $T(n/2, k) + T(n/2, m-k)$ gegeben, wobei $k \in [0 : m]$. Somit erhalten wir folgende Rekursionsformel für den Zeitbedarf:

$$\begin{aligned} T(n,m) &= 2c \cdot \frac{n}{2} \cdot m + cm + T\left(\frac{n}{2}, k\right) + T\left(\frac{n}{2}, m-k\right) \\ &= cnm + cm + T\left(\frac{n}{2}, k\right) + T\left(\frac{n}{2}, m-k\right) \end{aligned}$$

Wir könnten diese Rekursionsgleichung mit aufwendigen Mitteln direkt lösen. Wir machen es uns aber hier etwas leichter und verifizieren eine geratene Lösung mittels Induktion.

Behauptung: Es gibt eine Konstante $c \in \mathbb{R}_+$, so dass $T(n,m) \leq 2cnm + cm\log(n)$.

Induktionsanfang ($n = 1$): $T(1, m)$ ist sicherlich $O(m)$, da wir nur ein Zeichen gegen eine Zeichenreihe der Länge m optimal ausrichten müssen. Wenn wir c in der Behauptung hinreichend groß gewählt haben, so gilt die Behauptung sicherlich.

Induktionsschritt ($\to n$): Wir setzen nun die Behauptung als Induktionsvoraussetzung in die Rekursionsformel ein (da $\lceil n/2 \rceil < n$ für $n \geq 2$) und formen um:

$$\begin{aligned} T(n,m) &= cnm + cm + T\left(\frac{n}{2}, k\right) + T\left(\frac{n}{2}, m-k\right) \\ &\leq cnm + cm + 2c \cdot \frac{n}{2} \cdot k + ck\log\left(\frac{n}{2}\right) + 2c \cdot \frac{n}{2} \cdot (m-k) + c(m-k)\log\left(\frac{n}{2}\right) \\ &= cnm + cm + cnk + cn(m-k) + ck\log\left(\frac{n}{2}\right) + c(m-k)\log\left(\frac{n}{2}\right) \\ &= cnm + cm + cnm + cm\log\left(\frac{n}{2}\right) \\ &= 2cnm + cm + cm\log(n) - cm\log(2) \\ & \quad \text{da mit log der Logarithmus zur Basis 2 gemeint ist, gilt } \log(2) = 1 \\ &= 2cnm + cm + cm\log(n) - cm \\ &= 2cnm + cm\log(n). \end{aligned}$$

Damit ist die Laufzeit weiterhin $O(nm)$, da $m\log(n) \leq mn$ ist, und wir haben den folgenden Satz bewiesen.

Theorem 3.22 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$. Der Algorithmus von Hirschberg berechnet ein optimales globales paarweises Sequenzen Alignment für s und t in Zeit $O(nm)$ mit Platzbedarf $O(\min\{n, m\})$.*

Wir haben zwar nur einen Platzbedarf von $O(m)$ gezeigt, aber es sollte klar sein, dass man auch die Sequenzen s und t vertauschen kann, so dass die kürzere der beiden Sequenzen im Wesentlichen den benötigten Platzbedarf impliziert.

3.3 Besondere Berücksichtigung von Lücken

In diesem Abschnitt wollen wir teilweise Alignments und Strafen für Lücken genauer untersuchen. Eine Lücke ist nichts anderes als eine aufeinander folgende Folge von Edit-Operationen, die entweder nur aus Deletionen oder nur aus Insertionen bestehen (jedoch nicht abwechselnd). In einem Alignment entspricht dies einem Teilwort, das nur aus Leerzeichen $-$ besteht. Solche zusammenhängenden Lücken der Länge ℓ haben ihre Ursache meist aus einer einzigen Mutation, die eine ganze Teilsequenz entfernt bzw. eingefügt hat. Aus diesem Grund ist eine Bestrafung, die proportional zur Länge der Lücke ist, nicht ganz gerecht, und sollte daher eher sublinear in der Länge der Lücke sein.

3.3.1 Semi-Globale Alignments

Im Falle *semi-globaler Alignments* wollen wir Lücken, die am Anfang oder am Ende eines Wortes auftreten, nicht berücksichtigen. Dies ist insbesondere dann von Interesse, wenn die Wörter sehr unterschiedlich lang sind oder wenn klar ist, dass diese Sequenzen zwar eine Ähnlichkeit besitzen, aber man nicht weiß, ob man die Sequenzen korrekt aus einer großen Sequenz herausgeschnitten hat. Dann können an den Enden Fehler aufgetreten sein (etwas zu kurze oder zu lange Sequenzen gewählt).

Beispiel: Betrachten wir die beiden Sequenzen $CGTACGTGATGA$ und $CGATTA$. Wenn wir hierfür die optimale Alignment-Distanz berechnen (mit $w(x, y) = 3$ für $x \neq y \in \Sigma$ und $w(x, -) = 2$ für $x \in \Sigma$), so erhalten wir das folgende optimale Alignment:

$$\begin{array}{ccccccccccccc} C & G & T & A & C & G & T & G & A & G & T & G & A \\ C & G & - & A & - & - & T & - & - & - & T & - & A \end{array}$$

Dieses hat einen Alignment-Abstand von $7 * 2 = 14$.

Alternativ betrachten wir folgendes Alignment:

$$\begin{array}{cccccccccccccc} C & G & T & A & C & G & - & T & G & A & G & T & G & A \\ - & - & - & - & C & G & A & T & T & A & - & - & - & - \end{array}$$

Dieses hat natürlich eine größere Alignment-Distanz von $9 * 2 + 1 * 3 = 21$.

Berücksichtigen wir jedoch die Deletionen am Anfang und Ende nicht, da diese vermutlich nur aus einer zu lang ausgewählten ersten (oder zu kurz ausgewählten zweiten) Sequenz herrühren, so erhalten wir eine Alignment-Distanz von $1*2+1*3=5$. Aus diesem Grund werden bei einem semi-globalen Alignment Folgen von Insertionen bzw. Deletionen zu Beginn oder am Ende nicht berücksichtigt.

Es gibt jedoch noch ein kleines Problem. Man kann nämlich dann immer eine Alignment mit Alignment-Distanz 0 basteln:

$$\begin{array}{cccccccccccccccccc} C & G & T & A & C & G & T & G & A & G & T & G & A & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - & - & - & - & - & C & G & A & T & T \end{array}$$

Bei solchen Distanzen sollte man natürlich den Wert der Distanz im Verhältnis zur Länge des Bereiches in Beziehung setzen, in dem das eigentliche, bewertete Alignment steht. Man kann jetzt die Distanz bezüglich der wirklich ausgerichteten Zeichen um jeweils einen konstanten Betrag erniedrigen. Wir können uns das Leben jedoch viel einfacher machen, wenn wir statt dessen Ähnlichkeitsmaße verwenden. Wie wir schon gesehen haben, entsprechen diese im Wesentlichen den Distanzmaßen, sind aber bei solchen semi-globalen Alignments wesentlich einfacher zu handhaben.

Wir verwenden jetzt als Kostenfunktion für ein Ähnlichkeitsmaß für Matches $+1$, für Insertionen sowie Deletionen -1 und für Substitutionen -2. Dieses Kostenmaß ist aus der Kostenfunktion für das obige Distanzmaß mittels $1 - w(x, y)$ gewonnen worden. Somit erhält man für das erste globale Alignment einen Score von

$$6 * (+1) + 7 * (-1) = -1.$$

Für das zweite Alignment erhält man als globales Alignment einen Ähnlichkeitswert von

$$4 * (+1) + 8 * (-1) + 1 * (-2) = -6$$

und als semi-globales-Alignment einen Score von

$$4 * (+1) + 1 * (-1) + 1 * (-2) = +1.$$

Für das künstliche Alignment jedoch einen Score von 0. Wir weisen an dieser Stelle darauf hin, dass die hier verwendeten Kostenfunktionen nicht besonders gut gewählt, aber für die Beispiele ausreichend sind.

Wie äußert sich jetzt die Nichtberücksichtigung von Lücken am Anfang und Ende eines Alignments in der Berechnung dieser mit Hilfe der Dynamischen Programmierung nach Needleman-Wunsch. Betrachten wir zuerst noch einmal Abbildung 3.20, in der schematisch semi-globale Alignments dargestellt sind.

Wenn in der ersten Sequenz s am Anfang Lücken auftreten dürfen, bedeutet dies, dass wir in der zweiten Sequenz t Einfügungen gemacht haben. Damit diese nicht

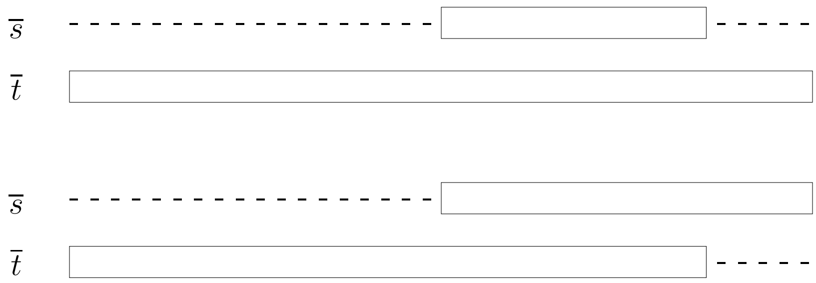


Abbildung 3.20: Skizze: semi-globale Alignments

zählen, dürfen diese Einfügungen zu Beginn nicht gewertet werden. Daher werden wir die erste Zeile der Tabelle mit 0 initialisieren. Analoges gilt für Lücken zu Beginn von t. Dann dürfen die Deletionen von s nicht bewertet werden und wir initialisieren auch die erste Spalte mit 0.

Nun betrachten wir Lücken am Ende. Tritt am Ende von s eine Lücke auf, dann dürfen wir die letzten Insertionen von Zeichen in t nicht berücksichtigen. Wie können wir dies bewerkstelligen? Dazu betrachten wir die letzte Zeile der Tabelle. Wenn die letzten Insertionen nicht zählen sollen, dann hört ein solches semi-globales Alignment irgendwo in der letzten Zeile auf. Wenn wir nun ein semi-globales Alignment mit maximaler Ähnlichkeit wollen, müssen wir einfach nur in der letzten Zeile den maximalen Wert suchen. Die Spalten dahinter können wir für unser semi-globales Alignment dann einfach vergessen.

Dasselbe gilt für Deletionen in s. Dann hört das semi-globale Alignment irgendwo in der letzten Spalte auf und wir bestimmen für ein optimales semi-globales Alignment den maximalen Wert in der letzten Spalte und vergessen die Zeilen danach.

In der folgenden Abbildung 3.21 sind die Pfade solcher semi-globaler Alignments in der berechneten Tabelle bildlich dargestellt.

Um also insgesamt ein optimales semi-globales Alignment zu erhalten, setzen wir die erste Zeile und erste Spalte gleich 0 und bestimmen den maximalen Ähnlichkeitswert, der in der letzten Zeile oder Spalte auftritt. Dieser gibt dann die Ähnlichkeit an. Das Alignment selbst erhalten wir dann genauso wie im Falle des globalen Alignments, indem wir einfach von diesem Maximalwert rückwärts das Alignment bestimmen. Wir hören auf, sobald wir die auf die erste Spalte oder die erste Zeile treffen.

Damit ergibt sich für die Tabelle S (wie Similarity):

$$S(i,j) = \begin{cases} 0 & \text{für } (i=0) \vee (j=0), \\ \max \left\{ \begin{array}{lll} S(i-1,j-1) & + & w(s_i,t_j), \\ S(i-1,j) & + & w(s_i,-), \\ S(i,j-1) & + & w(-,t_j) \end{array} \right\} & \text{für } (i>0) \wedge (j>0). \end{cases}$$

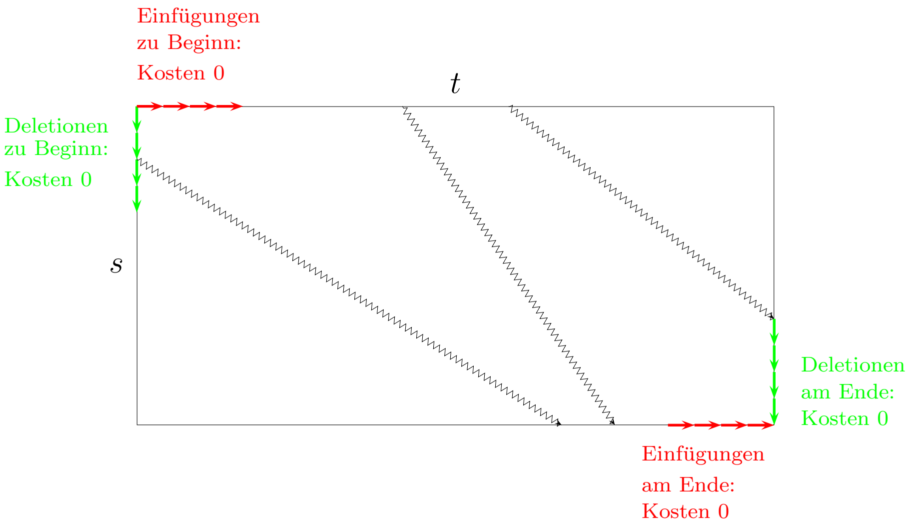


Abbildung 3.21: Skizze: Semi-Globale Alignments in der Ähnlichkeits-Tabelle

Natürlich lässt sich auch für semi-globale Alignments das Verfahren von Hirschberg zur Platzreduktion anwenden. Die Details seien dem Leser als Übungsaufgabe überlassen. Fassen wir unser Ergebnis noch zusammen.

Theorem 3.23 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$. Ein optimales semi-globales paarweises Sequenzen Alignment für s und t lässt sich in Zeit $O(nm)$ mit Platzbedarf $O(\min\{n, m\})$ berechnen.*

3.3.2 Lokale Alignments (Smith-Waterman)

Eine weitere Einschränkung sind so genannte *lokale Alignments*. Hier suchen wir in zwei Sequenzen zwei Teilwörter, die möglichst ähnlich zueinander sind. Damit wir nicht wieder zwei leere Teilwörter mit Alignment-Distanz 0 bekommen, verwenden wir auch hier wieder Ähnlichkeitsmaße. In der Abbildung 3.22 ist ein solches lokales Alignment schematisch dargestellt.

Abbildung 3.22: Skizze: lokales Alignment

Betrachten wir zunächst ein Beispiel, nämlich ein lokales Alignment zwischen den Sequenzen $s = ACGATTATTT$ und $t = TAGTAATCG$, wie es in Abbildung 3.23 dargestellt ist. Das lokales Alignment besteht aus den beiden Teilwörtern, die in dem grauen Rahmen eingefasst sind, und hat den Ähnlichkeitswert 7 (hierbei ist $w(x, x) = 3$, $w(x, y) = -3$ und $w(x, -) = -2$ für $x \neq y \in \Sigma$).

s:	A	C	G	A	T	T	A	T	T	T
t:	T	A	G	-	T	A	A	T	C	G

Abbildung 3.23: Beispiel: Ein lokales Alignment zwischen s und t

Wie können wir nun ein solches lokales Alignment berechnen? Wir werden auch hier die Methode von Needleman-Wunsch wiederverwenden. Dazu definieren wir $S(i, j)$ als den Wert eines besten lokalen Alignments von zwei Teilwörtern von $s_1 \cdots s_i$ und $t_1 \cdots t_j$. Dann können wir wieder eine Rekursionsgleichung aufstellen:

$$S(i,j) = \begin{cases} 0 & \text{für } (i = 0) \vee (j = 0), \\ \max \left\{ \begin{array}{lcl} S(i-1, j-1) & + & w(s_i, t_j), \\ S(i-1, j) & + & w(s_i, -), \\ S(i, j-1) & + & w(-, t_j), \\ & 0 & \end{array} \right\} & \text{für } (i > 0) \wedge (j > 0). \end{cases}$$

Die Rekursionsgleichung sieht fast so aus, wie im Falle der semi-globalen Alignments. Wir müssen hier nur in der Maximumsbildung im Falle von $i \neq 0 \neq j$ den Wert 0 berücksichtigen. Das folgt daraus, dass ein lokales Alignment ja an jeder Stelle innerhalb der beiden gegebenen Sequenzen i und j beginnen kann. Wie finden wir nun ein optimales lokales Alignment? Da ein lokales Alignment ja an jeder Stelle innerhalb der Sequenzen s und t enden darf, müssen wir einfach nur den maximalen Wert innerhalb der Tabelle S finden. Dies ist dann der Ähnlichkeitswert eines optimalen lokalen Alignments.

Das Alignment selbst finden wir dann wieder durch Rückwärtsverfolgen der Sieger aus der Maximumbildung. Ist der Sieger letztendlich der Wert 0 in der Maximumsbildung, so haben wir den Anfangspunkt eines optimalen lokalen Alignments gefunden. Die auf dieser Rekursionsgleichung basierende Methode wird oft auch als Algorithmus von Smith-Waterman bezeichnet.

In der folgenden Abbildung 3.24 ist noch einmal der Pfad, der zu einem lokalen Alignment gehört, innerhalb der Tabelle S schematisch dargestellt.

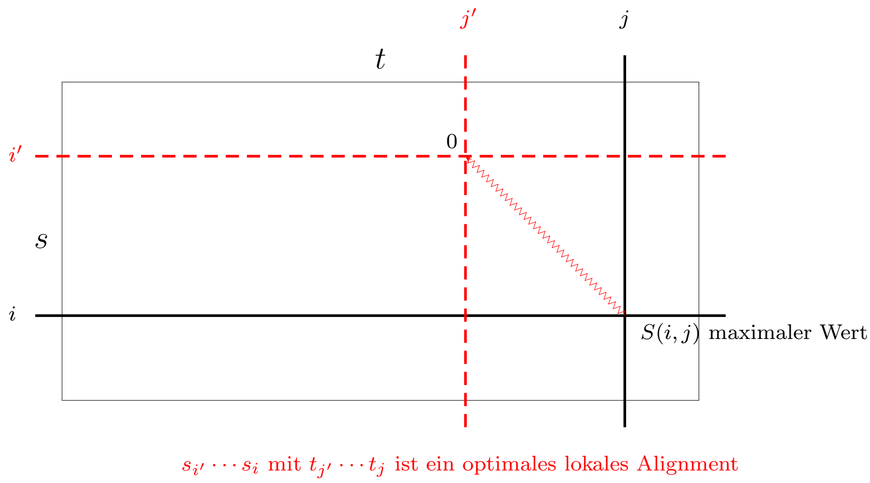


Abbildung 3.24: Skizze: lokales Alignment in der Tabelle

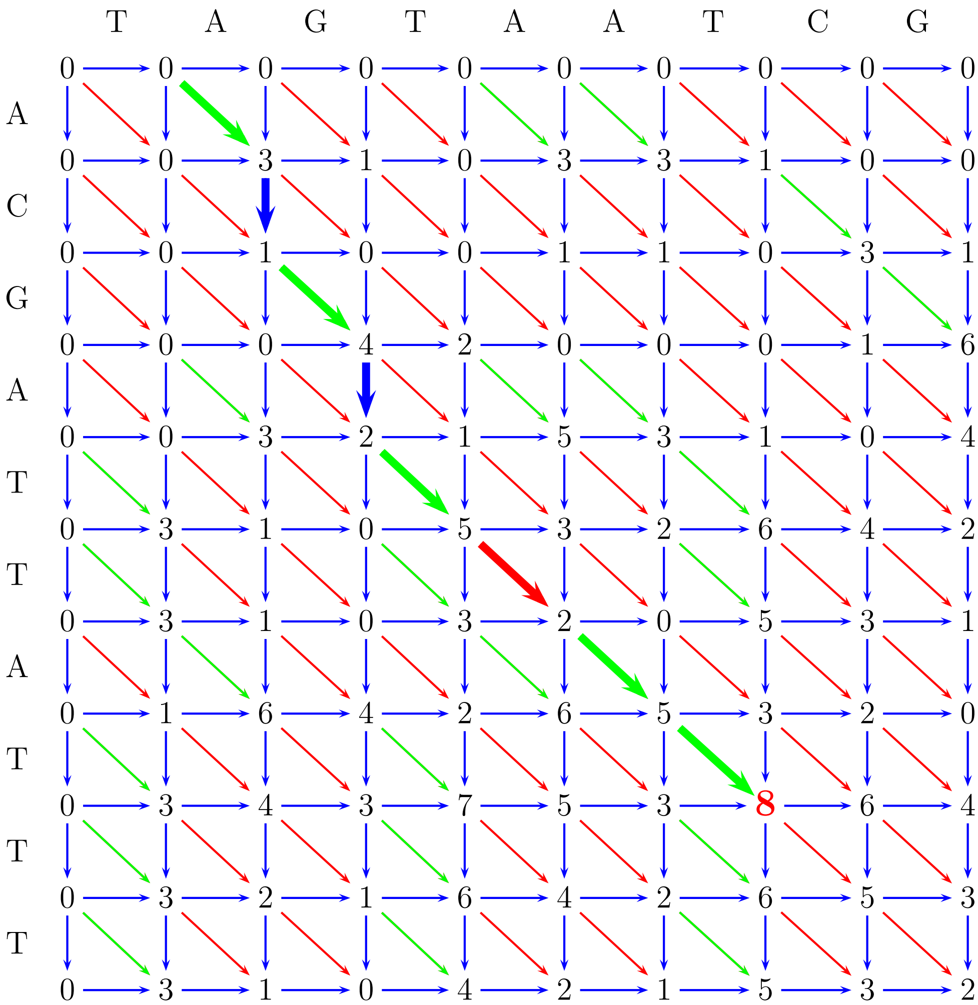
Auch für das lokale paarweise Sequenzen Alignment lässt sich die Methode von Hirschberg zur Platzreduktion anwenden. Wir überlassen es wieder dem Leser sich die genauen Details zu überlegen. Zusammenfassend erhalten wir für lokale Alignments das folgende Ergebnis.

Theorem 3.24 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$. Ein optimales lokales paarweises Sequenzen Alignment für s und t sowie der zugehörige Ähnlichkeitswert lässt sich in Zeit $O(nm)$ mit Platzbedarf $O(\min\{n, m\})$ berechnen.*

Kehren wir noch einmal zu unserem konkreten Beispiel vom Anfang des Abschnitts zurück und berechnen die Tabelle S für die Sequenzen $s = ACGATTATTT$ und $t = TAGTAATCG$. Die Tabelle mit den zugehörigen Werten ist in Abbildung 3.25 angegeben.

Wie man leicht sieht ist der maximale Wert 8 (siehe Position $(8, 7)$ in der Tabelle für S). Der zurückverfolgte Weg für das optimale lokale Alignment ist in der Abbildung durch die dicken Pfeile dargestellt.

Auf die Null trifft man an der Position $(0, 1)$ in der Tabelle S. Aus diesem Pfad lässt sich wie üblich wieder das zugehörige lokale Alignment ablesen. Dies ist explizit in der Abbildung 3.26 angegeben. Das zu Beginn angegebene lokale Alignment war also nicht optimal, aber schon ziemlich nahe dran. Durch eine Verlängerung kommt man auf das optimale lokale Alignment.



Match = +3, Indel = −2, Subst = −3

Abbildung 3.25: Beispiel:Tabelle für lokale Alignments zwischen s und t

s:	-	A	C	G	A	T	T	A	T	T	T
t:	T	A	-	G	-	T	A	A	T	C	G

Abbildung 3.26: Beispiel: Ein optimales lokales Alignment zwischen s und t

3.3.3 Lücken-Strafen

Manchmal tauchen in Alignments mittendrin immer wieder lange Lücken auf (siehe Abbildung 3.27). Eine Lücke der Länge ℓ nun mit den Kosten von ℓ Insertionen oder Deletionen zu belasten ist nicht unbedingt fair, da diese durch eine Mutation entstanden sein kann. Dass kurze Lücken wahrscheinlicher als lange Lücken sind, ist noch einzusehen. Daher sollte die Strafe monoton in der Länge sein.

Lücke

T A T A _ _ _ _ A G C T C T C G

T A T A C C G C A G C _ _ _ C G

Lücke

Abbildung 3.27: Skizze: Lücken in Alignments

Zur Bestrafung für Lücken verwenden wir eine Lücken-Strafe (engl. gap-penalty), die durch eine Funktion $g : \mathbb{N}_0 \to \mathbb{R}$ gegeben ist. Hierbei gibt $g(k)$ die Strafe für k konsekutive Insertionen bzw. Deletionen an. Im Falle von Distanzmaßen ist g immer nichtnegativ und im Falle von Ähnlichkeitsmaßen nichtpositiv. Dabei sollte immer $g(0) = 0$ gelten und $|g| : \mathbb{N}_0 \to \mathbb{R}_+ : k \mapsto |g(k)|$ eine monoton wachsende Funktion sein. Außerdem nehmen wir an, dass die Lücken-Strafe g sublinear ist, d.h. $g(k' + k'') \leq g(k') + g(k'')$ für alle $k', k'' \in \mathbb{N}_0$. Wir bemerken hier noch, dass wir jetzt Insertionen und Deletionen explizit immer gleich bewerten, unabhängig davon, welche Zeichen gelöscht oder eingefügt werden. In Abbildung 3.28 ist skizziert, wie Funktionen für „vernünftige“ Lücken-Strafen aussehen. Lineare Strafen haben wir bereits berücksichtigt, da ja die betrachteten Distanz- und Ähnlichkeitsmaße linear waren. Im nächsten Abschnitt beschäftigen wir uns mit beliebigen Lückenstrafen, dann mit affinen und zum Schluss geben wir noch einen Ausblick auf konkave Lückenstrafen.

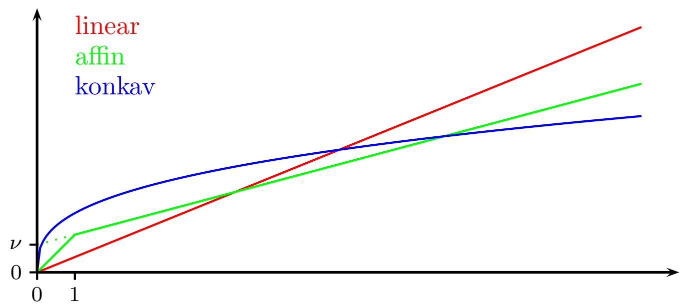


Abbildung 3.28: Skizze: Funktionsgraphen einiger typischer Lücken-Strafen

3.3.4 Allgemeine Lücken-Strafen (Waterman-Smith-Byers)

Nun wollen wir uns damit beschäftigen, wie wir die Rekursionsgleichungen für Alignments für allgemeine Lückenstrafen anpassen können. Wir beschränken uns hier wieder auf Distanzmaße und globale Alignments. Die Übertragung auf Ähnlichkeitsmaße und nichtglobale Alignments sei dem Leser zur Übung überlassen

Für allgemeine Lücken-Strafen ergeben sich die folgenden Rekursionsgleichungen nach Waterman-Smith-Byers.

$$D(i,j) = \begin{cases} g(i) & \text{für } i = 0, \\ g(j) & \text{für } j = 0, \\ \min_k \left\{ \begin{array}{lcl} D(i-1,j-1) & + & w(s_i,t_j), \\ D(i-k,j) & + & g(k), \\ D(i,j-k) & + & g(k) \end{array} \right\} & \text{für } (i>0) \wedge (j>0). \end{cases}$$

Im Gegensatz zum Algorithmus von Needleman-Wunsch muss hier bei der Aktualisierung von $D(i,j)$ auf alle Werte in derselben Zeile bzw. Spalte bei Insertionen und Deletionen zurückgegriffen werden, da die Kosten der Lücken ja nicht linear sind und somit nur im ganzen und nicht einzeln berechnet werden können. Im Prinzip werden hier auch zwei unmittelbar aufeinander folgende Lücken berücksichtigt, da aber die Strafe von zwei unmittelbar aufeinander folgenden Lücken der Länge k' und k'' größer als die einer Lücke der Länge $k'+k''$ ist dies kein Problem. Wir haben hierbei ausgenutzt, dass g sublinear ist, d.h. $g(k'+k'') \leq g(k') + g(k'')$ für alle $k', k'' \in \mathbb{N}_0$.

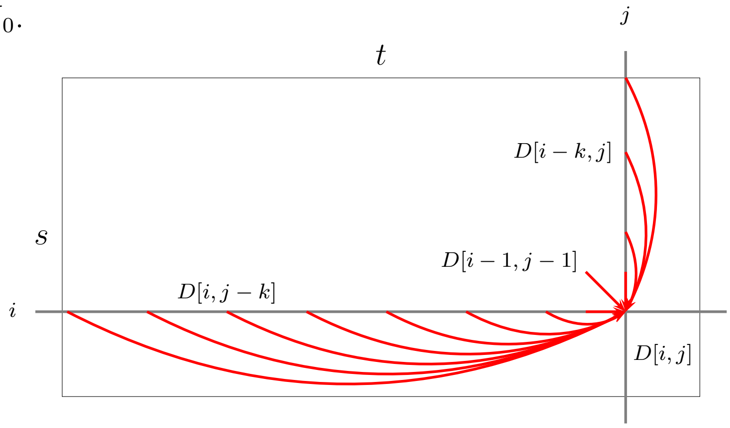


Abbildung 3.29: Skizze: Berechnung optimaler Alignments nach Waterman-Smith-Byers

In der Abbildung 3.29 ist noch einmal schematisch dargestellt, auf welche Werte die Berechnung von $D[i,j]$ zurückgreift.

Die Laufzeit für die Variante von Waterman-Smith-Byers ist jetzt größer geworden, da für jeden Tabellen-Eintrag eine Minimumbildung von $O(n+m)$ Elementen involviert ist. Damit wird die Laufzeit im Wesentlichen kubisch nämlich $O(nm(n+m))$. Fassen wir das Ergebnis zusammen.

Theorem 3.25 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$. Ein optimales globales paarweises Sequenzen Alignment für s und t mit allgemeinen Lücken-Strafen lässt sich in Zeit $O(nm(n + m))$ mit Platzbedarf $O(nm)$ berechnen.*

Leider lässt sich hier die Methode von Hirschberg nicht anwenden, da zur Bestimmung optimaler Alignment-Distanzen, alle vorherigen Zeilen benötigt werden.

3.3.5 Affine Lücken-Strafen (Gotoh)

Da für allgemeine Lücken-Strafen sowohl Laufzeit- als auch Platzbedarf zu hoch sind, schauen wir uns spezielle Lücken-Strafen an, nämlich affine Lücken-Strafen. Solche affinen Lücken-Strafen lassen sich wie folgt beschreiben:

$$g : \mathbb{N} \to \mathbb{R}_+ : k \mapsto \mu \cdot k + \nu$$

für Konstanten $\mu, \nu \in \mathbb{R}_+$. Für $g(0)$ setzen wir, wie zu Beginn gefordert $g(0) = 0$, so dass nur die Funktion auf $\mathbb{N}$ im bekannten Sinne affin ist. Dennoch werden wir solche Funktionen für eine Lücken-Strafe affin nennen. Hierbei sind ν die Kosten, die für das Auftauchen einer Lücke prinzipiell berechnet werden (so genannte *Strafe für Lückeneröffnung*), und μ die proportionalen Kosten für die Länge der Lücke (so genannte *Strafe für Lückenfortsetzung*).

Wieder können wir eine Rekursionsgleichung zur Berechnung optimaler Alignment-Distanzen angeben. Der daraus resultierende Algorithmus wird der Algorithmus von Gotoh genannt. Die Rekursionsgleichungen sind etwas komplizierter, insbesondere deswegen, da wir jetzt vier Tabellen berechnen müssen, die wie folgt definiert sind:

- $E[i, j]$ = Distanz eines optimalen Alignments von $s_1 \cdots s_i$ mit $t_1 \cdots t_j$, das mit einer Einfügung endet.
- $F[i, j]$ = Distanz eines optimalen Alignments von $s_1 \cdots s_i$ mit $t_1 \cdots t_j$, das mit einer Löschung endet.
- $G[i, j]$ = Distanz eines optimalen Alignments von $s_1 \cdots s_i$ mit $t_1 \cdots t_j$, das mit einer Substitution endet.
- $D[i, j]$ = Distanz eines optimalen Alignments von $s_1 \cdots s_i$ mit $t_1 \cdots t_j$.

Letztendlich ist man natürlich nur an der Tabelle D interessiert, zu deren Berechnung jedoch die anderen Tabellen benötigt werden. Die Rekursionsgleichungen ergeben sich wie folgt:

- Betrachten wir zuerst die Tabelle E, d.h. das Alignment endet mit einer Insertion. Dann muss davor eine Substitution oder eine Insertion gewesen sein, da aufgrund der Dreiecksungleichung eine Insertion nicht auf eine Deletion folgen kann. Im ersten Fall wird eine Lücke eröffnet (Kosten $\nu + \mu$), im anderen Fall eine fortgesetzt (Kosten μ). Somit erhalten wir:

$$E[i, j] = \min\{G[i, j-1] + \mu + \nu, E[i, j-1] + \mu\}.$$

Dies ist in der folgenden Abbildung noch einmal schematisch dargestellt.

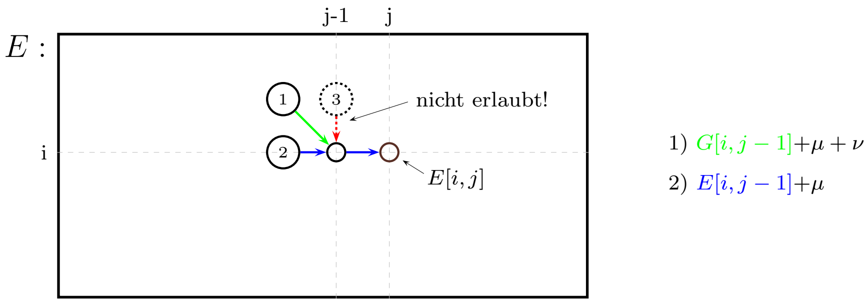


Abbildung 3.30: Skizze: Erweiterung eines Alignments mit einer Insertion

- Betrachten wir jetzt die Tabelle F, d.h. das Alignment endet mit einer Deletion. Dann muss davor eine Substitution oder eine Deletion gewesen sein, da aufgrund der Dreiecksungleichung eine Deletion nicht auf eine Insertion folgen kann. Im ersten Fall wird eine Lücke eröffnet (Kosten $\nu + \mu$), im anderen Fall eine fortgesetzt (Kosten μ). Somit erhalten wir:

$$F[i, j] = \min\{G[i-1, j] + \mu + \nu, F[i-1, j] + \mu\}.$$

Dies ist in der folgenden Abbildung noch einmal schematisch dargestellt.

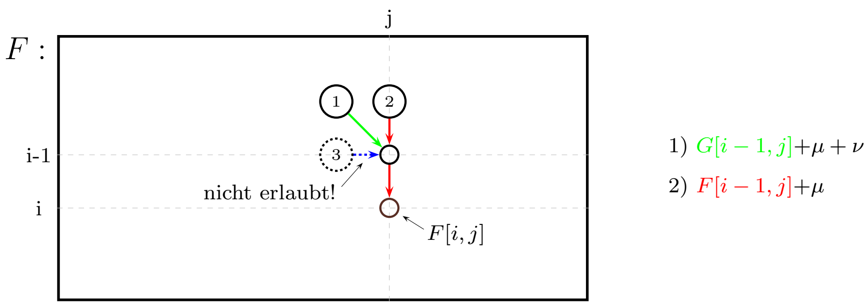


Abbildung 3.31: Skizze: Erweiterung eines Alignments mit einer Deletion

- Betrachten wir jetzt die Tabelle G, d.h. das Alignment endet mit einer Substitution. Wir müssen nur berücksichtigen, ob das Alignment zuvor mit einer Substitution, Deletion oder Insertion geendet hat. Dann erhalten wir:

$$G[i, j] = \min \left\{ \begin{array}{l} G[i-1, j-1] + w(s_i, t_j), \\ E[i-1, j-1] + w(s_i, t_j), \\ F[i-1, j-1] + w(s_i, t_j) \end{array} \right\}.$$

Dies ist in der folgenden Abbildung noch einmal schematisch dargestellt.

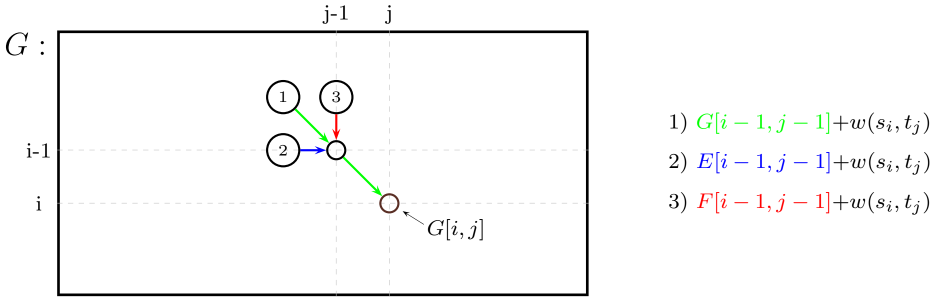


Abbildung 3.32: Skizze: Erweiterung eines Alignments mit einer Substitution

- Die Tabelle D berechnet sich offensichtlich aus dem Minimum aller drei Tabellen
$$D[i, j] = \min\{E[i, j], F[i, j], G[i, j]\}.$$

Bei Ähnlichkeitsmaßen sehen die Rekursionsgleichungen im Wesentlichen gleich aus, es wird nur die Minimumsbildung durch eine Maximumsbildung ersetzt und im Falle der Tabellen E und F müssen auch Deletionen und Insertionen berücksichtigt werden, da bei Ähnlichkeitsmaßen aufgrund der fehlenden Dreiecksungleichung auch Insertionen und Deletionen unmittelbar benachbart sein dürfen.

Es stellt sich nun noch die Frage, welche Werte jeweils in der 1. Zeile bzw. in der 1. Spalte der Matrizen stehen. Es gilt für $i > 0$ und $j > 0$:

$$\begin{aligned}
E[0, j] &= j * \mu + \nu \\
E[i, 0] &= \infty \\
E[0, 0] &= \infty \\
\\
F[i, 0] &= i * \mu + \nu \\
F[0, j] &= \infty \\
F[0, j] &= \infty \\
\\
G[i, 0] &= \infty \\
G[0, j] &= \infty \\
G[0, 0] &= 0
\end{aligned}$$

Auch hier kann man wieder die Methode von Hirschberg zur Platzreduktion anwenden. Halten wir noch das Ergebnis fest.

Theorem 3.26 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$. Ein optimales globales paarweises Sequenzen Alignment für s und t mit affinen Lücken-Strafen lässt sich in Zeit $O(nm)$ mit Platzbedarf $O(\min(n, m))$ berechnen.*

3.3.6 Konkave Lücken-Strafen

Zum Abschuss wollen wir noch kurz konkave Lücken-Strafen erwähnen. Eine Funktion $f : \mathbb{N}_0 \to \mathbb{R}$ heißt *konkav*, wenn gilt:

$$\forall n \in \mathbb{N} : \; f(x) - f(x-1) \leq f(x+1) - f(x).$$

Anschaulich bedeutet dies, dass die Funktion immer langsamer wächst. In der kontinuierlichen Analysis ist dies gleichbedeutend damit, dass die erste Ableitung monoton fallend ist bzw. die zweite Ableitung kleiner gleich Null ist (natürlich nur sofern die Funktion zweimal differenzierbar ist). Ein bekannter Vertreter von konkaven Funktionen ist der Logarithmus.

Wir wollen an dieser Stelle nicht näher auf konkave Funktionen als Lücken-Strafen eingehen, sondern nur ein Ergebnis festhalten.

Theorem 3.27 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$. Ein optimales globales paarweises Sequenzen Alignment für s und t mit konkaven Lücken-Strafen lässt sich in Zeit $O(nm \log(n))$ berechnen.*

Wir merken noch an, dass gewisse konkave Funktionen die Berechnung sogar in Zeit $O(nm)$ zulassen.

3.4 Hybride Verfahren

In diesem Abschnitt wollen wir uns mit so genannten hybriden Verfahren beschäftigen. Dies sind Verfahren die mehrere verschiedene Techniken gleichzeitig einsetzen. Hier wird es eine Alignment-Berechnung mit Hilfe von Suffix-Bäumen sein.

3.4.1 One-Against-All-Problem

Im *One-Against-All-Problem* wollen wir für zwei gegebene Sequenzen $s, t \in \Sigma^*$ alle globalen Alignments von s gegen alle Teilwörter von t berechnen. Formal wird das Problem wie folgt beschrieben.

Geg.: $s, t \in \Sigma^*$ mit $|s| = n$, $|t| = m$.
Ges.: Berechne $d(s, t')$ für alle $t' \sqsubseteq t$.

Hierbei gilt $t' \sqsubseteq t$ für ein gegebenes $t \in \Sigma^*$, wenn t' ein Teilwort von t ist.

Wir betrachten zuerst einen naiven Ansatz und berechnen für jedes Teilwort t' von t dessen Alignment gegen s. Die Anzahl der Teilwörter von t mit $|t| = m$ beträgt $\Theta(m^2)$. Da der Aufwand pro Alignment $O(nm)$ ist, ist der Gesamtaufwand $O(nm^3)$.

Etwas geschickter können wir vorgehen, wenn wir uns an die Tabelle $D(i, j)$ erinnern und bemerken, dass wir ja nicht nur die optimale Alignment-Distanz $s = s_1 \cdots s_n$ mit $t = t_1 \cdots t_m$ berechnen, sondern auch gleich für alle Paare s mit $t_1 \cdots t_j$ für $j \in [0 : m]$. Diese Distanzen stehen in der letzten Zeile. Somit brauchen wir die Distanzen nur für alle Suffixe $t^k := t_k \cdots t_m$ von t mit s zu berechnen. Wir können dann die Ergebnisse der Distanzen von $t_k \cdots t_\ell$ für $k \leq \ell \in [0 : m]$ mit s auslesen. da es nur $O(m)$ Suffixe von t gibt ist der Zeitbedarf dann nur noch $O(nm^2)$.

Wir können noch ein wenig effizienter werden, wenn wir die Suffixe von t mit Hilfe eines Suffix-Baumes T_t verwalten. Wir durchlaufen jetzt diesen Suffix-Baum mit Hilfe der Tiefensuche. Für jeden Knoten, den wir besuchen, erhalten wir ein Suffix von t und berechnen für dieses die Tabelle der optimalen Alignment-Distanzen gegen s.

Hierbei können wir einige Einträge jedoch geschickt recyceln. Betrachten wir zwei Knoten v und w die durch eine Kante $(v, w) \in T_t$ verbunden sind (w ist also ein Kind von v) und die beiden zugehörigen Suffixe t_v und t_w von t. Ist $\text{label}(v, w)$ das Kantenlabel der Kante (v, w), dann gilt nach Definition eines Suffix-Baumes: $t_w = t_v \cdot \text{label}(v, w)$. Um nun die Tabelle für s und t_w zu berechnen, können wir die linke Hälfte für s und t_v wiederverwenden und müssen nur die letzten $|\text{label}(v, w)|$ Spalten neu berechnen. Dies ist schematisch in Abbildung 3.33 dargestellt.

Damit ergibt sich unmittelbar der folgende Algorithmus, der in Abbildung 3.34 angegeben ist. Hierbei berechnet $\text{compute\_table}(D, s, t', k, \ell)$ die Tabelle mit den optimalen Alignment-Distanzen von s mit $t' = t_1 \cdots t_\ell$, wobei die Einträge $D(i, j)$ für $i \in [0 : n]$ und $j \in [0 : k]$ schon berechnet sind.

Die Laufzeit beträgt dann $O(m)$ für den reinen DFS-Durchlauf, da der Suffix-Baum ja $O(|t|) = O(m)$ Knoten besitzt. Für die Berechnung der Tabellenerweiterungen

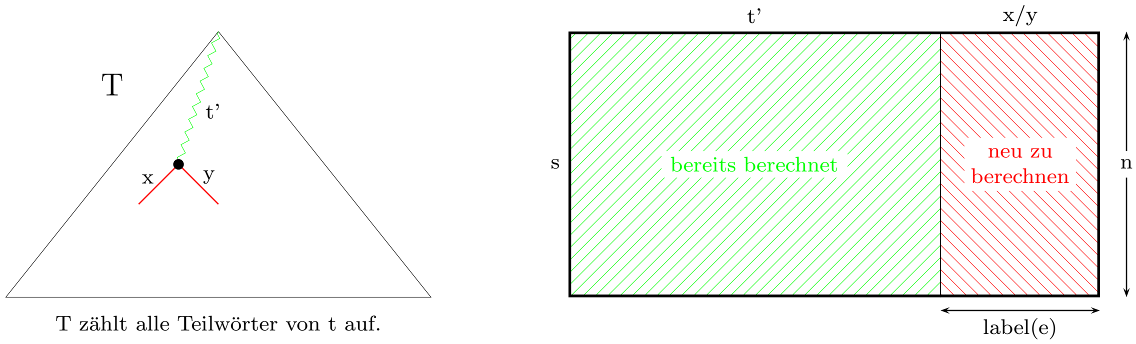


Abbildung 3.33: Skizze: Hybrides Verfahren für One-Against-All

fällt für jede Kante $(v, w) \in E(T_t)$ die Zeit $O(|s| \cdot |\text{label}(v, w)|)$ an. Somit ergibt sich für die gesamte Laufzeit $T(n, m)$:

$$\begin{aligned} T(n,m) &= O(m) + O\left(\sum_{(v,w)\in E(T_t)} n \cdot |\text{label}(v,w)| \right) \\ &= O(m) + O\left(n \cdot \underbrace{\sum_{(v,w)\in E(T_t)} |\text{label}(v,w)|}_{=:\text{size}(\text{T}_t)} \right) \\ &= O(m + n \cdot \text{size}(T_t)). \end{aligned}$$

Hierbei ist die Größe size eines Suffixbaumes durch die Summe aller Längen der Kantenlabels gegeben. Wie wir uns bei den Suffix-Tries schon überlegt haben, gilt dann für einen Suffix-Baum T_t: $\text{size}(T_t) = O(m^2) \cap \Omega(m)$.

Theorem 3.28 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$. Alle optimalen Alignment-Distanzen für s und t' mit $t \sqsubseteq t$ lassen sich mit Hilfe des hybriden Verfahrens in Zeit $O(n \cdot \text{size}(T_t)) \subseteq O(nm^2)$ bestimmen.*

DFS (node v, char $s[]$, char $t[]$)
{
  for all $((v, w) \in E(T_t))$ **do**
  {
    compute_table($D, s, t' \cdot \text{label}(v, w), |t'|, |t' \cdot \text{label}(v, w)|$);
    DFS($w, s, t \cdot \text{label}(v, w)$);
  }
}

Abbildung 3.34: Algorithmus: Hybrides Verfahren für One-Against-All

Experimente mit realistischen (biologischen) Daten haben ergeben, dass $size(T_t)$ in der Regel ungefähr $m^2/10$ entspricht.

3.4.2 All-Against-All-Problem

Im *All-Against-All-Problem* wollen wir für zwei gegebene Sequenzen $s, t \in \Sigma^*$ alle globalen Alignments von s' gegen alle Teilwörter von t berechnen, sofern diese Distanz ein gewisse Schranke ϑ unterschreitet. Formal wird das Problem wie folgt beschrieben.

Geg.: $s, t \in \Sigma^*$ mit $|s| = n$ sowie $|t| = m$ und $\vartheta \in \mathbb{R}_+$.
Ges.: Berechne $d(s', t')$ für alle $s' \sqsubseteq s$ und $t' \sqsubseteq t$ mit $d(s', t') \leq \vartheta$.

Wir betrachten zuerst einen naiven Ansatz und berechnen für jedes Teilwort s' von s sowie jedes Teilwort t' von t deren Alignment. Die Anzahl der Teilwörter von s' bzw. t' von s bzw. t mit $|s| = n$ bzw. $|t| = m$ beträgt $\Theta(n^2m^2)$. Da der Aufwand pro Alignment $O(nm)$ ist, beträgt der Gesamtaufwand $O(n^3m^3)$.

Etwas geschickter können wir wieder vorgehen, wenn wir uns an die Tabelle $D(i, j)$ erinnern und bemerken, dass wir ja nicht nur die optimale Alignment-Distanz von $s = s_1 \cdots s_n$ mit $t = t_1 \cdots t_m$ berechnen, sondern auch gleich für alle Paare $s_1 \cdots s_i$ mit $t_1 \cdots t_j$ für $i \in [0 : n]$ und $j \in [0 : m]$. Diese Distanzen stehen ja über die gesamte Tabelle verteilt in $D(i, j)$. Somit brauchen wir die Distanzen nur für alle Suffixe $s^k = s_k \cdots s_n$ und $t^{k'} := t_{k'} \cdots t_m$ von t mit s zu berechnen. Wir können dann die Ergebnisse der Distanzen von $s_k \cdots s_\ell$ und $t_{k'} \cdots t_{\ell'}$ für $k \leq \ell \in [0 : n]$ und $k' \leq \ell' \in [0 : m]$ aus D auslesen. Da es nur $O(n)$ Suffixe von s und $O(m)$ Suffixe von t gibt, ist der Zeitbedarf dann nur noch $O(n^2m^2)$.

Ist $\vartheta = \infty$ so ist diese Methode optimal, da wir ja $\Theta(n^2m^2)$ Paare von Ergebnissen ausgeben müssen. Da wir jetzt nur noch die Paare ausgeben wollen, deren Alignment-Distanz kleiner gleich ϑ ist, können wir mit Hilfe eines hybriden Verfahrens effizienter vorgehen.

Wir werden wieder die Suffixe von s und t mit Hilfe von Suffix-Bäumen verwalten. Hierzu sei T_s bzw. T_t der Suffix-Baum von s bzw. t. Wir durchlaufen jetzt beide Suffix-Bäume von s und t parallel mit Hilfe der Tiefensuche. Für jedes Paar von Knoten $(v, v') \in V(T_s) \times V(T_t)$ die wir besuchen, erhalten wir ein Paar von Suffixen von s' bzw. t' und berechnen für diese die Tabelle der optimalen Alignment-Distanzen.

Hierbei können wir wiederum einige Einträge geschickt recyceln. Betrachten wir zwei Paare von Knoten v und w sowie v' und w', die durch eine Kante $(v, w) \in E(T_s)$ sowie $(v', w') \in E(T_t)$ verbunden sind (w bzw. w' ist also ein Kind von v bzw. v') und die beiden zugehörigen Suffixe s_v von s und $t_{v'}$ von t. Ist label(v, w) bzw.

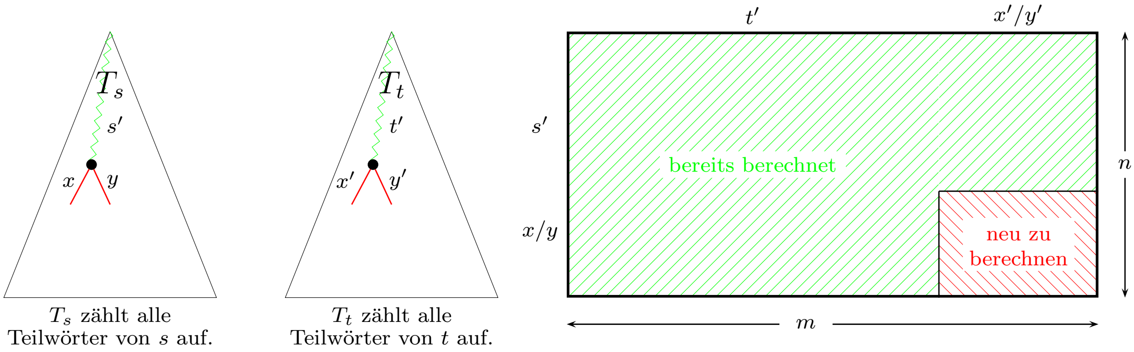


Abbildung 3.35: Skizze: Hybrides Verfahren für All-Against-All

label(v', w') das Kantenlabel der Kante (v, w) bzw. (v', w'), dann gilt nach Definition eines Suffix-Baumes: $s_w = s_v \cdot \text{label}(v, w)$ bzw. $t_{w'} = t_{v'} \cdot \text{label}(v', w')$. Um nun die Tabelle für s_w und $t_{w'}$ zu berechnen, können wir den größten Teil links und oben für s_v bzw. s_w sowie $t_{v'}$ wiederverwenden und müssen nur den rechten unteren Teil ($|\text{label}(v, w)|$ Zeilen sowie $|\text{label}(v', w')|$ Spalten) neu berechnen. Dies ist schematisch in Abbildung 3.35 dargestellt.

Damit ergibt sich der in Abbildung 3.36 angegebene Algorithmus, der zu Beginn mit $\text{DFS}_\text{s}(r(T_s), \varepsilon)$ aufgerufen wird. Die Prozedur compute_table$(D, s', t', k, k', \ell, \ell')$ berechnet die Tabelle mit den optimalen Alignment-Distanzen von $s' = s_1 \cdots s_\ell$ mit $t' = t_1 \cdots t_{\ell'}$, wobei einige Einträge in $D(i, j)$ schon berechnet sind.

Für die Laufzeit $T(n, m)$ gilt (wobei der Term $O(nm)$ vom reinen parallelen Durchlaufen der Suffix-Bäume mit der Tiefensuche herrührt):

$$\begin{aligned}
T(n,m) &= O(nm) + O\left(\sum_{(v,w)\in E(T_s)} \sum_{(v',w')\in E(T_t)} |\text{label}(v,w)| \cdot |\text{label}(v',w')| \right) \\
&= O(nm) + O\left(\sum_{(v,w)\in E(T_s)} |\text{label}(v,w)| \sum_{(v',w')\in E(T_t)} |\text{label}(v',w')| \right) \\
&= O(n+m) + O(\text{size}(T_s) \cdot \text{size}(T_t)).
\end{aligned}$$

Theorem 3.29 *Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$ sowie $\vartheta > 0$. Alle optimalen Alignment-Distanzen für s' und t' mit $s' \sqsubseteq s$ und $t' \sqsubseteq t$ und $d(s', t') \leq \vartheta$ lassen sich mit Hilfe des hybriden Verfahrens in Zeit $O(\text{size}(T_s) \cdot \text{size}(T_t) + D)$ bestimmen, wobei D die Anzahl der Paare (s', t') mit $d(s', t') \leq \vartheta$ ist.*

DFS_S (node v, char s[])
{
  for all $((v, w) \in E(T_s))$ **do**
  {
    DFS_t($r(T_t)$, $s \cdot \text{label}(v, w)$, ε, $|s|$, $|s \cdot \text{label}(v, w)|$)
    DFS_s(w, $s \cdot \text{label}(v, w)$)
  }
}

DFS_T (node w, char s[], char t[], int k, int k')
{
  for all $((w, w;) \in E(T_t))$ **do**
  {
    compute_table($D, s, t \cdot \text{label}(v', w'), k, k', |t|, |t \cdot \text{label}(v', w')|$);
    DFS_t($w', s, t \cdot \text{label}(v', w')$);
  }
}

Abbildung 3.36: Algorithmus: Hybrides Verfahren für All-Against-All

3.5 Datenbanksuche

In diesem Abschnitt wollen wir noch kurz die verwendeten algorithmischen Ideen der beiden gebräuchlichsten Tools zur Suche in großen Sequenz-Datenbanken vorstellen: FASTA und BLAST. Wir stellen hier nur jeweils die Hauptvariante vor. Von beiden Verfahren gibt es zahlreiche abgeleitete Varianten.

3.5.1 FASTA (FAST All oder FAST Alignments)

Im Folgenden suchen wir nach einer Sequenz s in einer Datenbank t.

(1) Wir wählen zuerst eine Konstante ktup in Abhängigkeit vom Inhalt der Datenbank:
$$k := \text{ktup} = \begin{cases} 6 & \text{für DNS} \\ 2 & \text{für Proteine} \end{cases}$$
Dann suchen wir nach perfekten Treffern von Teilwörtern von s der Länge k in t, d.h. für solche Treffer (i, j) gilt $s_i \cdots s_{i+k-1} = t_j \cdots t_{j+k-1}$. Dies erfolgt mit Hilfe einer Hash-Tabelle entweder für die Datenbank oder für das Suchmuster. Da es nur wenige kurze Sequenzen gibt ($4^6 = 4096$ bei DNS und $20^2 = 400$ bei

Proteinen), kann man für jede solche kurze Sequenz eine Liste mit den zugehörigen Positionen in t speichern, an der solche kurzen Sequenz auftreten. Diese kurzen Treffer von $s_i \cdots s_{i+k-1}$ werden *Hot Spots* genannt. Diese Hot Spots sind

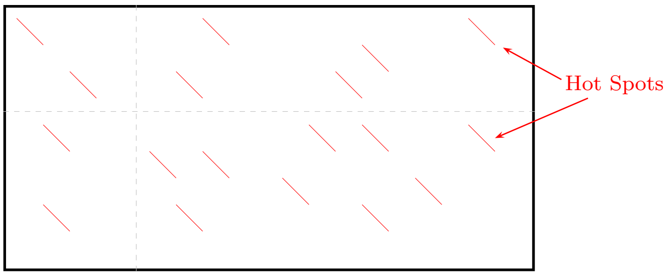


Abbildung 3.37: Skizze: Hot Spots

in der Abbildung 3.37 noch einmal in der (nicht wirklich berechneten) Tabelle für die Alignment-Distanzen visualisiert.

(2) Jetzt werden auf den Diagonalen der Tabelle mit den Alignment-Distanzen (wiederum ohne diese explizit zu berechnen) so genannte *Diagonal Runs* gesucht. Das sind mehrere Hot Spots die sich in derselben Diagonalen befinden, so dass die Lücken dazwischen kurz sind. Dies ist in Abbildung 3.38 noch einmal illustriert. Dazu bewertet man die Hot-Spots positiv und die Lücken negativ, wobei

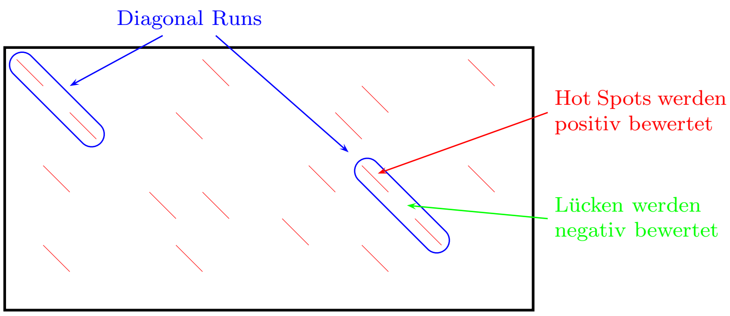


Abbildung 3.38: Skizze: Diagonal Runs

längere Lücken einen kleineren (negativen!) Wert erhalten als kurze Lücken. Wir bewerten nun die Folgen von Hot Spots in ihren Diagonalen, ähnlich wie bei einem lokalen Alignment. Die etwa zehn besten werden zum Schluss aufgesammelt. Wir merken hier noch an, dass nicht alle Hot Spots einer Diagonalen in einem Diagonal Run zusammengefasst werden müssen und dass es in einer Diagonalen durchaus mehr als einen Diagonal Run geben kann.

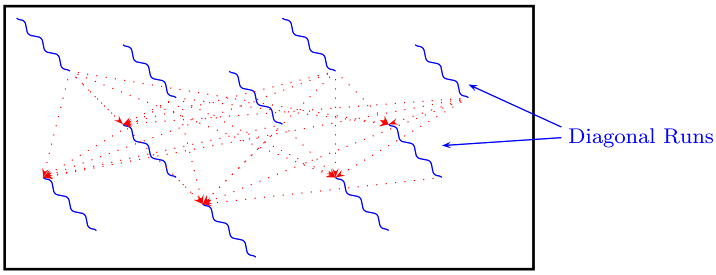


Abbildung 3.39: Skizze: Graph aus Diagonal Runs

(3) Nun erzeugen wir einen gerichteten Graphen. Die Knoten entsprechen den Diagonal Runs aus dem vorherigen Schritt und erhalten die positiven Gewichte, die im vorhergehenden Schritt bestimmt wurden. Zwei Diagonal Runs werden mit einer Kante verbunden, wenn der Endpunkt des ersten Diagonal Runs oberhalb des Anfangspunktes des zweiten Diagonal Runs liegt. Die Kanten erhalten wiederum ein negatives Gewicht, das entweder konstant oder proportional zum Abstand der Endpunkte ist. Der Graph ist noch einmal in Abbildung 3.39 illustriert. Der so entstandene Graph ist azyklisch (d.h. kreisfrei) und wir können darin wieder sehr einfach gewichtsmaximale Pfade suchen.

(4) Für die gewichtsmaximalen Pfade aus Diagonal Runs berechnen wir jetzt noch ein semiglobales Alignment. Da wir nur an kleinen Distanzen interessiert sind, brauchen wir nur kleine Umgebungen dieser Pfade von Diagonal Runs zu berücksichtigen, was zu einer linearen Laufzeit (in $|s|$) führt. Dies ist in Abbildung 3.40 bildlich dargestellt.

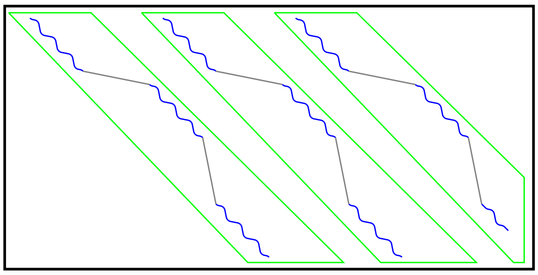


Abbildung 3.40: Skizze: Optimale Alignments um die Pfade aus Diagonal Runs

3.5.2 BLAST (Basic Local Alignment Search Tool)

Wieder nehmen wir an, dass wir nach einer Sequenz s in einer Datenbank t suchen.

(1) Zuerst konstruieren wir alle Wörter aus Σ^k und testen, ob für das verwendete Ähnlichkeitsmaß S gilt: $S(s_i \cdots s_{i+k-1}, w) \geq \vartheta$. Ist dies der Fall, so nehmen wir dieses Wort in eine Suchmustermenge M auf Hierbei wird k relativ klein gewählt:
$$k \begin{cases} \in [3:5] & \text{für Proteine,} \\ \approx 12 & \text{für DNS.} \end{cases}$$
Diese Menge M beinhaltet nun Wörter, die ziemlich ähnlich zu Teilwörtern aus dem ursprünglichen Suchmuster s sind. Der Vorteil ist der, dass wir die Fehler jetzt extrahiert haben und im Weiteren mit einer exakten Suche weitermachen können.

(2) Jetzt suchen wir in der Datenbank t nach Wörtern aus M, z.B. mit Hilfe des Algorithmus von *Aho-Corasick*, und merken uns die Treffer.

(3) Sei $j \in [1:m]$ mit $t_j \cdots t_{j+k-1} = w \in M$ und $s_i \cdots s_{i+k-1}$ das Teilwort s' aus s ist, für den $S(s', w)$ maximal wurde; s' ist also ein Zeuge dafür, dass w in M aufgenommen wurde. Jetzt berechnen wir den Wert $S(s_i \cdots s_{i+k-1}, t_j \cdots t_{j+k-1})$. Ist dieser Ähnlichkeitswert größer als ϑ, so nennen wir $(s_i \cdots s_{i+k-1}, t_j \cdots t_{j+k-1})$ ein *Sequence Pair*.

(4) Solche Sequence Pairs sind Startwerte für mögliche gute lokale Alignments von s und t. Zum Schluss erweitern wir Sequence Pairs zu einem optimalen lokalen Alignment und geben diese als Treffer zusammen mit dem erzielten Score aus.

3.6 Konstruktion von Ähnlichkeitsmaßen

In diesem Abschnitt wollen wir einen kurzen Einblick geben, wie man aus experimentellen biologischen Daten gute Kostenfunktionen für Ähnlichkeitsmaße konstruieren kann.

3.6.1 Maximum-Likelihood-Prinzip

Zuerst erläutern wir kurz das so genannte *Maximum-Likelihood-Prinzip*, dass hinter der Entscheidung für ein bestimmtes Modell aufgrund experimenteller Daten steckt.

Definition 3.30 (Maximum-Likelihood-Prinzip) *Gibt es mehrere Modelle zur Erklärung eines Sachverhalts (experimentell ermittelte Daten), so wählt man das Modell, das für die ermittelten Daten die größte Wahrscheinlichkeit vorhersagt.*

Für das Problem des Sequenzen Alignments kann man sich zwei simple Modelle vorstellen. Das erste ist das so genannte *Zufallsmodell* R. Hier nehmen wir an dass zwei ausgerichtete Sequenzen gar nichts miteinander zu tun haben und gewisse Übereinstimmungen rein zufällig sind.

Für die Wahrscheinlichkeit für das Auftreten eines Alignments (s, t) für $s, t \in \Sigma^n$ gilt dann in diesem Modell:

$$\mathrm{Ws}((s,t) \mid R) = \prod_{i=1}^{n} p_{s_i} \prod_{i=1}^{n} p_{t_i}.$$

Hierbei ist p_a die Wahrscheinlichkeit, dass in einer Sequenz das Zeichen $a \in \Sigma$ auftritt. Wir haben für diese Alignments jedoch angenommen, dass Leerzeichen nicht erlaubt sind (also keine Deletionen und Insertionen, sondern nur Substitutionen).

Ein anderes Modell ist das so genannte *Mutationsmodell* M, wobei wir annehmen, dass ein Alignment (s, t) für $s, t \in \Sigma^n$ durchaus erklärbar ist, nämlich mit Hilfe von Mutationen. Hier gilt für die Wahrscheinlichkeit für ein Alignment (s, t)

$$\mathrm{Ws}((s,t) \mid M) = \prod_{i=1}^{n} (p_{s_i} \cdot p_{s_i,t_i}).$$

Hierbei bezeichnet $p_{a,b}$ die Wahrscheinlichkeit, dass in einer Sequenz das Symbol a zu einem Symbol b mutiert. Wir nehmen an, dass $p_{a,b} = p_{b,a}$ gilt.

Vergleichen wir jetzt beide Modelle, d.h. wir dividieren die Wahrscheinlichkeiten für ein gegebenes Alignment (s, t) mit $s, t \in \Sigma^n$:

$$\frac{\mathrm{Ws}((s,t) \mid M)}{\mathrm{Ws}((s,t) \mid R)} = \prod_{i=1}^{n} \frac{p_{s_i} \cdot p_{s_i,t_i}}{p_{s_i} \cdot p_{t_i}} \lesseqgtr 1.$$

Ist nun dieser Bruch größer als 1, so spricht diese für das Mutationsmodell, andernfalls beschreibt das Zufallsmodell dieses Alignment besser.

Leider wäre dieses Maß multiplikativ und nicht additiv. Da können wir uns jedoch sehr einfach mit einem arithmetischen Trick behelfen. Wir logarithmieren die Werte:

$$\mathrm{Score}(s,t) := \sum_{i=1}^{n} \underbrace{\log\left(\frac{p_{s_i} \cdot p_{s_i,t_i}}{p_{s_i} \cdot p_{t_i}}\right)}_{Kostenfunktion}$$

Aus diesem Ähnlichkeitsmaß können wir nun eine zugehörige Kostenfunktion für alle Paare $(a, b) \in \Sigma \times \Sigma$ sehr leicht ableiten, nämlich den logarithmierten Quotienten der einzelnen Wahrscheinlichkeiten, dass sich ein solches Paar innerhalb eines Alignments gegenübersteht:

$$w(a, b) := \log \left(\frac{p_a \cdot p_{a,b}}{p_a \cdot p_b} \right).$$

Es bleibt die Frage, wie man p_a bzw. $p_{a,b}$ für $a, b \in \Sigma$ erhält?

3.6.2 PAM-Matrizen

In diesem Abschnitt wollen für die obige Frage eine Lösung angeben. Wir nehmen hierzu an, wir erhalten eine Liste von so genannten *akzeptierten Mutationen* $L = (\{a_1, b_1\}, \ldots, \{a_n, b_n\})$, d.h. wir können sicher sein, dass die hier vorgekommenen Mutationen wirklich passiert sind. Solche Listen kann man über mehrfache Sequenzen Alignments von gut konservierten Regionen ähnlicher Spezies erhalten. Mit $n_{a,b}$ bezeichnen wir die Paare $\{a, b\}$ in der Liste L und mit n die Anzahl aller Paare in L.

Für p_a mit $a \in \Sigma$ ist es am einfachsten, wenn man hierfür die relative Häufigkeit von a in allen Sequenzen annimmt. Es bleibt insbesondere $p_{a,b}$ zu bestimmen. Die Mutation $a \to b$ ist nichts anderes, als die bedingte Wahrscheinlichkeit, dass in einer Sequenz ein b auftritt, wo vor der Mutation ein a stand. Für diese bedingte Wahrscheinlichkeit schreiben wir $\mathrm{Ws}(b \mid a)$. Nach dem Satz von Bayes für bedingte Wahrscheinlichkeiten gilt, dass $\mathrm{Ws}(b \mid a) = \frac{\mathrm{Ws}(a,b)}{\mathrm{Ws}(a)}$, wobei $\mathrm{Ws}(a, b)$ die Wahrscheinlichkeit ist, dass einem Alignment a und b gegenüberstehen. Also gilt:

$$p_{a,b} = \mathrm{Ws}(b \mid a) = \frac{\mathrm{Ws}(a, b)}{\mathrm{Ws}(a)} \sim \frac{\frac{n_{a,b}}{n}}{p_a} = \frac{n_{a,b}}{n} \cdot \frac{1}{p_a}.$$

Die letzte Proportionalität folgt daher, dass wir für die Wahrscheinlichkeit $\mathrm{Ws}(a, b)$ annehmen, dass diese durch die relative Häufigkeit von Mutationen ziemlich gut angenähert wird (bis auf einen konstanten Faktor). Da in unserer Liste L nur Mutationen stehen, wissen wir natürlich nicht, mit welcher Wahrscheinlichkeit eine Mutation wirklich auftritt. Daher setzen wir zunächst etwas willkürlich für $a \neq b$ an:

$$\begin{aligned} p_{a,b} &:= \frac{n_{a,b}}{n} \cdot \frac{1}{p_a} \cdot \frac{1}{100}, \\ p_{a,a} &:= 1 - \sum_{\substack{b \in \Sigma \\ b \neq a}} p_{a,b}. \end{aligned}$$

Zunächst gilt für alle $a \in \Sigma$:

$$\sum_{b \in \Sigma} p_{a,b} = p_{a,a} + \sum_{\substack{b \in \Sigma \\ b \neq a}} p_{a,b} = 1 - \sum_{\substack{b \in \Sigma \\ b \neq a}} p_{a,b} \cdot + \sum_{\substack{b \in \Sigma \\ b \neq a}} p_{a,b} = 1.$$

Da außerdem nach Definition $p_{a,b} \in [0,1]$ gilt, handelt es sich um eine zulässige Wahrscheinlichkeitsverteilung. Weiter gilt

$$\begin{aligned}
\sum_{a \in \Sigma} p_a \cdot p_{a,a} &= \sum_{a \in \Sigma} p_a \left(1 - \sum_{\substack{b \in \Sigma \\ b \neq a}} p_{a,b}\right) \\
&= \underbrace{\sum_{a \in \Sigma} p_a}_{=1} - \sum_{a \in \Sigma} \sum_{\substack{b \in \Sigma \\ b \neq a}} p_a \cdot p_{a,b} \\
&= 1 - \sum_{a \in \Sigma} \sum_{\substack{b \in \Sigma \\ b \neq a}} \frac{n_{ab}}{n} \cdot \frac{1}{p_a} \cdot \frac{1}{100} \cdot p_a \\
&= 1 - \frac{1}{100n} \cdot \underbrace{\sum_{a \in \Sigma} \sum_{\substack{b \in \Sigma \\ b \neq a}} n_{ab}}_{=n} \\
&= 0{,}99.
\end{aligned}$$

Somit gilt, dass mit Wahrscheinlichkeit 99% keine Mutation auftritt und mit Wahrscheinlichkeit 1% eine Mutation auftritt. Aus diesem Grund werden diese Matrizen $(p_{a,b})_{a,b \in \Sigma}$ auch *1-PAM* genannt. Hierbei steht PAM für *Percent Accepted Mutations* oder *Point Accepted Mutations*. Als zugehörige *Kostenfunktion* erhalten wir dann

$$w(a,b) = \log\left(\frac{p_a \cdot p_{a,b}}{p_a \cdot b}\right) = \log\left(\frac{p_a \cdot \frac{1}{p_a} \cdot \frac{n_{ab}}{n} \cdot \frac{1}{100}}{p_a \cdot p_b}\right) = \log\left(\frac{n_{ab}}{100 \cdot n \cdot p_a \cdot p_b}\right)$$

Diese PAM-Matrizen wurden erfolgreich für kleine evolutionäre Abstände von Margaret Dayhoff auf der Basis von Aminosäuren entwickelt und eingesetzt.

Diese 1-PAM Matrizen sind jetzt jedoch nur für sehr kurze evolutionäre Abstände geeignet. Man kann diese jedoch auch auf so genannte k-PAM-Matrizen hochskalieren, indem man die Matrix $P = (p_{a,b})_{a,b \in \Sigma}$ durch $P^k = (p^{(k)}_{a,b})_{a,b \in \Sigma}$ ersetzt und dann entsprechend in die Kostenfunktion einsetzt. Diese Methode liefert zum Beispiel so genannte 120- oder 250-PAM-Matrizen, die dann für größere evolutionäre Abstände einsetzbar sind. Für wirklich große evolutionäre Abstände haben sich jedoch PAM-Matrizen als nicht so brauchbar erwiesen. Hier werden dann meist so genannte BLOSUM-Matrizen eingesetzt, auf die wir an dieser Stelle jedoch nicht eingehen wollen.

Literaturhinweise

A.1 Lehrbücher zur Vorlesung

Peter Clote, Rolf Backofen: *Introduction to Computational Biology*; John Wiley and Sons, 2000.

Richard Durbin, Sean Eddy, Anders Krogh, Graeme Mitchison: *Biological Sequence Analysis*; Cambridge University Press, 1998.

Dan Gusfield: *Algorithms on Strings, Trees, and Sequences — Computer Science and Computational Biology*; Cambridge University Press, 1997.

David W. Mount: *Bioinformatics — Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, 2001.

Pavel A. Pevzner: *Computational Molecular Biology - An Algorithmic Approach*; MIT Press, 2000.

João Carlos Setubal, João Meidanis: *Introduction to Computational Molecular Biology*; PWS Publishing Company, 1997.

Michael S. Waterman: *Introduction to Computational Biology: Maps, Sequences, and Genomes*; Chapman and Hall, 1995.

A.2 Skripten anderer Universitäten

Bonnie Berger: *Introduction to Computational Molecular Biology*, Massachusetts Institute of Technology, http://theory.lcs.mit.edu/~bab/01-18.417-home.html;

Bonnie Berger, *Topics in Computational Molecular Biology*, Massachusetts Institute of Technology, Spring 2001, http://theory.lcs.mit.edu/~bab/01-18.418-home.html;

Paul Fischer: *Einführung in die Bioinformatik* Universität Dortmund, Lehrstuhl II, WS2001/2002, http://ls2-www.cs.uni-dortmund.de/lehre/winter200102/bioinf/

Richard Karp, Larry Ruzzo: *Algorithms in Molecular Biology*; CSE 590BI, University of Washington, Winter 1998. http://www.cs.washington.edu/education/courses/590bi/98wi/

Larry Ruzzo: *Computational Biology*, CSE 527, University of Washington, Fall 2001; http://www.cs.washington.edu/education/courses/527/01au/

Georg Schnittger: *Algorithmen der Bioinformatik*, Johann Wolfgang Goethe-Universität Frankfurt am Main, Theoretische Informatik, WS 2000/2001, http://www.thi.informatik.uni-frankfurt.de/BIO/skript2.ps.

Ron Shamir: *Algorithms in Molecular Biology* Tel Aviv University, http://www.math.tau.ac.il/~rshamir/algmb.html; http://www.math.tau.ac.il/~rshamir/algmb/01/algmb01.html.

Ron Shamir: *Analysis of Gene Expression Data, DNA Chips and Gene Networks*, Tel Aviv University, 2002; http://www.math.tau.ac.il/~rshamir/ge/02/ge02.html;

Martin Tompa: *Computational Biology*, CSE 527, University of Washington, Winter 2000. http://www.cs.washington.edu/education/courses/527/00wi/

A.3 Lehrbücher zu angrenzenden Themen

Teresa K. Attwood, David J. Parry-Smith; *Introduction to Bioinformatics*; Prentice Hall, 1999.

Maxime Crochemore, Wojciech Rytter: *Text Algorithms*; Oxford University Press: New York, Oxford, 1994.

Martin C. Golumbic: *Algorithmic Graph Theory and perfect Graphs*; Academic Press, 1980.

Benjamin Lewin: *Genes*; Oxford University Press, 2000.

Milton B. Ormerod: *Struktur und Eigenschaften chemischer Verbindungen*; Verlag Chemie, 1976.

Hooman H. Rashidi, Lukas K. Bühler: *Grundriss der Bioinformatik — Anwendungen in den Biowissenschaften und der Medizin*,

Klaus Simon: *Effiziente Algorithmen für perfekte Graphen*; Teubner, 1992.

Maxine Singer, Paul Berg: *Gene und Genome*; Spektrum Akademischer Verlag, 2000.

Lubert Stryer: *Biochemie*, Spektrum Akademischer Verlag, 4. Auflage, 1996.

A.4 Originalarbeiten

Kellogg S. Booth, George S. Lueker: Testing for the Consecutive Ones property, Interval Graphs, and Graph Planarity Using PS-Tree Algorithms; *Journal of Computer and System Science*, Vol.13, 335–379, 1976.

Ting Chen, Ming-Yang Kao: On the Informational Asymmetry Between Upper and Lower Bounds for Ultrametric Evolutionary Trees, *Proceedings of the 7th Annual European Symposium on Algorithms, ESA'99*, Lecture Notes in Computer Science 1643, 248–256, Springer-Verlag, 1999.

Richard Cole: Tight Bounds on the Complexity of the Boyer-Moore String Matching Algorithm; *SIAM Journal on Computing*, Vol. 23, No. 5, 1075–1091, 1994.
s.a. *Technical Report*, Department of Computer Science, Courant Institute for Mathematical Sciences, New York University, TR1990-512, June, 1990, http://csdocs.cs.nyu.edu/Dienst/UI/2.0/Describe/ncstrl.nyu_cs%2fTR1990-512

Martin Farach, Sampath Kannan, Tandy Warnow: A Robust Model for Finding Optimal Evolutionary Trees, *Algorithmica*, Vol. 13, 155–179, 1995.

Wen-Lian Hsu: PC-Trees vs. PQ-Trees; *Proceedings of the 7th Annual International Conference on Computing and Combinatorics, COCOON 2001*, Lecture Notes in Computer Science 2108, 207–217, Springer-Verlag, 2001.

Wen-Lian Hsu: A Simple Test for the Consecutive Ones Property; *Journal of Algorithms*, Vol.43, No.1, 1–16, 2002.

Haim Kaplan, Ron Shamir: Bounded Degree Interval Sandwich Problems; *Algorithmica*, Vol. 24, 96–104, 1999.

Edward M. McCreight: A Space-Economical Suffix Tree Construction Algorithm; *Journal of the ACM*, Vol. 23, 262–272,1976.

Moritz Maaß: *Suffix Trees and Their Applications*, Ausarbeitung von der Ferienakademie '99, Kurs 2, Bäume: Algorithmik und Kombinatorik, 1999. http://www14.in.tum.de/konferenzen/Ferienakademie99/

Esko Ukkonen: On-Line Construction of Suffix Tress, *Algorithmica*, Vol. 14, 149–260, 1995.

Index

Symbole

A

B

F

G

H

P

Q

R