Alignment von DNA Sequenzen

Seminar Bio–Informatik; Institut für Informatik IV; Universität Bonn

Prof. Blum, C. Rick, Dr. Vingron, M. Nikolaidou

Olaf Schmidt*

27. April 1998

Zusammenfassung

Die Ähnlichkeiten von biomolekularen Sequenzen (DNA, RNA) haben funktionelle oder strukturelle Bedeutung. Dieser Seminarbeitrag behandelt anhand von Beispielen die grundlegenden Funktionen und Algorithmen, die Sequenzen analysieren und die Ähnlichkeiten berechnen.

Zunächst arbeitet man mit Editiersequenzen, die eine Sequenz in eine andere überführen. Diese Editiersequenz muß minimiert werden, um ein optimales Alignment zu finden. Eine andere Vorgehensweise definiert ein Ähnlichkeitsmaß zweier Sequenzen, das dann maximiert wird.

Neben dem Vergleich von ganzen Sequenzen werden auch Teilsequenzen betrachtet. Dazu sucht man kurze Sequenzen in einer langen oder auch ähnliche Teilsequenzen in zwei langen Folgen.

*Email: olaf@dodo.rhein.de

1 Einführung

Der Vergleich von Sequenzen ist heute ein unverzichtbarer Bestandteil der Molekularbiologie geworden. Damit verbunden ist das Speichern, Sortieren und Suchen in Datenbanken.

1.1 Biologische Grundlagen

Die DNA (Desoxyribonucleinsäure) ist aus einer Kette von Molekülen aufgebaut. Die Kettenglieder unterscheiden sich durch einen Basenrest, der funktionelle Bedeutung hat. Es gibt die Basenreste Adenin (A), Thymin (T), Guanin (G) und Cytosin (C). Die Ketten (oder Sequenzen) kodieren Proteinsequenzen. Proteine sind die Grundbausteine des Lebens. Um ein Protein zu erzeugen, werden Teilsequenzen der DNA mit Hilfe der viel kürzeren RNA (Ribonucleinsäure) Sequenzen kopiert und dann zu Proteinsequenzen wieder zusammengebaut. Die RNA bestehen aus Basen-Triplets und kodieren jeweils ein Kettenglied der Proteinsequenz. Die Ketten (oder Sequenzen) werden als Strings dargestellt (siehe Anhang).

Die *Ähnlichkeiten* von biomolekularen Sequenzen (DNA, RNA) haben funktionelle oder strukturelle Bedeutung. Die Evolution benutzt, verdoppelt und verändert erfolgreiche Strukturen, um sie neu zu kombinieren und zu verbessern. Das Leben basiert auf kleinen Bausteinen, die herumgereicht und geteilt werden. Die gleichen Bausteine findet man in verschiedenen Spezies immer wieder. Sie sind Grundbausteine des Lebens. Dabei entstehen — aus Kombinationen von alten — neue Bausteine, die neue Funktionen bieten.

Durch diese ständigen Duplikationen und Modifikationen entsteht eine große Menge von *Redundanzen*. Diese Redundanzen machen die Arbeit mit dem Vergleich von Sequenzen wertvoll. Die Sequenzen oder Teilsequenzen haben sich mit der Zeit verändert, sind aber immer noch ähnlich. Das zeigt, daß Objekte der Molekular-Biologie robust sind: Sie haben die Freiheit einige Variationen zu verkraften. Identifiziert man solche Objekte bei verschiedenen Spezies, so lassen sich Rückschlüsse auf den Verlauf der Evolution ziehen.

Die biomolekulare Analyse kann auf verschiedenen Ebenen stattfinden. Die DNA Sequenzen sind aus verschiedenen biochemischen Elementen aufgebaut. Teilsequenzen codieren Proteinsequenzen, die mit Hilfe von RNA zusammengebaut werden. Diese verschiedenen Level hängen biologisch zusammen und können in der Bio-Informatik als verschiedene Abstraktionsebenen betrachtet werden. So kann man auch Ähnlichkeiten in verschiedenen Leveln berechnen.

Um diese Sequenzvergleiche machen zu können und Ähnlichkeiten zu berechnen, werden zunächst große Datenbanken mit Sequenzen benötigt. Mit geeigneten Algorithmen wird dann in einer solchen Datenbank eine Probe-Sequenz mit den vorhandenen verglichen. Diese Algorithmen sind Thema dieses Seminarvortrages.

Eine neue Denkweise setzt sich in der Molekularbiologie durch. Sobald die Datenbanken ausreichend viel Material enthalten, kann man Hypothesen über funktionelle oder strukturelle

Zusammenhänge aufgrund von den gespeicherten Informationen aufstellen und direkt verifizieren. Experimente sind nur noch nötig, um die Effekte zu beweisen oder in der Realität zu demonstrieren.

1.2 Editiersequenzen

Bei den DNA Sequenzen entstehen neue Mutationen indem einzelne Teile der Basen-Ketten verändert werden. Dabei können einzelne Basen durch andere ersetzt werden (z.B. bei Fehl-Kopien), Basenreste wegfallen (z.B. durch Strahlung) und neue Basenreste hinzukommen (z.B. Einbaufehler).

Um zwei Sequenzen vergleichen zu können betrachtet man die Anzahl der Operationen, die man auf eine Sequenz anwenden muß, um diese in eine andere zu überführen. Die Operationen werden nacheinander angewendet; so werden Schritt für Schritt die Sequenzen überführt. Eine Folge von Editieroperationen nennt man *edit transcript*.

Es gibt vier mögliche Editieroperationen, die an jeder Stelle angewendet werden können:

R ein Zeichen wird ersetzt (*replace, substitute*)
M zwei Zeichen sind gleich (*match*)
I es wird ein Zeichen eingefügt (*insert*)
D es wird ein Zeichen gelöscht (*delete*)

Im folgenden Beispiel wurde der String ,,vintner" (Weinhändler) in ,,writers" (Schreiber) überführt:

```
RIMDMDMMI
v intner
wri t ers
```

In der ersten Zeile steht das Edit Transcript. Die einzelnen Editieroperationen überführen den String zeichenweise in den neuen String: Das v wurde durch ein w ersetzt. Ein r wurde eingefügt. Das folgende i stimmte in beiden Strings überein. Dann wurde ein n gelöscht. Das t ist wieder identisch und das folgende n wurde gelöscht. Es folgen zwei identische Zeichen e und r. Zum Schluß wurde noch das s eingefügt.

Definition Ein String über dem Alphabet R,M,I,D beschreibt die Transformation eines Strings in einen anderen. Dies nennt man das *edit transcript* zweier Strings.

1.3 Alignment

Hat man ein edit transcript auf eine Sequenz angewendet, so erhält man als Ergebnis ein Alignment. Die Sequenzen schreibt man als Strings von Zeichen untereinander, um sie direkt vergleichen zu können. Gelöschte und eingefügte Zeichen werden mit dem Platzhalter `-` markiert:

```
v-intner-
wri-t-ers
```

Definition Ein *globales Alignment* ist eine Gegenüberstellung zweier Strings. Dabei werden zunächst `-` Zeichen in die Strings eingebaut, und dann die Strings so übereinander geschrieben, daß zu jedem Zeichen oder `-` des ersten Strings genau ein Zeichen oder `-` des anderen Strings zugeordnet wird. Nur $\binom{-}{-}$ ist nicht erlaubt.

Man kann auch zwei Sequenzen insgesamt und nicht als Folge der Bearbeitung durch Editieroperationen betrachten. Dann gibt es drei String-Operationen:

1. Bei der *Substitution* wird eine Base durch eine andere ersetzt. In der String-Darstellung wird einfach der andere Buchstabe des Alphabets eingesetzt.

$$\begin{aligned} a &= \mathrm{AC}\boxed{\mathrm{T}}\mathrm{GC} \\ b &= \mathrm{AC}\boxed{\mathrm{C}}\mathrm{GC} \end{aligned}$$

2. Beim *Löschen* fällt ein Basenpaar weg. An die Stelle des Strings wird das Null-Zeichen „–" eingebaut.

$$\begin{aligned} b &= \mathrm{A}\boxed{\mathrm{C}}\mathrm{CGC} \\ c &= \mathrm{A}\boxed{-}\mathrm{CGC} \end{aligned}$$

3. Beim *Einfügen* von einer Base wird in der String-Darstellung im Originalstring ein Null-Zeichen eingebaut und an der gleichen Stelle im neuen String das zusätzliche Zeichen eingefügt.

$$\begin{aligned} c &= \mathrm{ACG}\boxed{-}\mathrm{C} \\ d &= \mathrm{ACG}\boxed{\mathrm{T}}\mathrm{C} \end{aligned} \tag{1}$$

Einfügen eines Zeichens in eine Sequenz kann auch als Löschen des gleichen Zeichens in der anderen Sequenz gesehen werden (und umgekehrt). Deshalb werden diese beiden Operationen zusammengefaßt (*indel*).

Das Alignment des o.g. Beispiels ist dann

ACTG–C
A–CGTC.

Es hat drei Identities, zwei Indels und zusätzlich ein nicht passendes Pärchen (mismatch).

Das Ziel von Sequenz Alignment ist es die Veränderung der Sequenzen darzustellen, ohne zu wissen welche einzelnen Operationen stattgefunden haben. Ohne das Wissen der Transitionen $a \to b \to c \to d$ würde man vermuten, daß das Alignment

ACTG–C
AC–GTC

der Transition entspricht ist, da es vier gleiche Zeichen (Identity) und zwei Veränderungen (Indels) hat.

1.4 Unterschiede

Das edit transcript und Alignment sind mathematisch gesehen äquivalent. Man kann sie ineinander überführen. Der Unterschied besteht in der Modellierung oder Sichtweise. Das edit transcript beschreibt den *Vorgang* der Mutation von einer Sequenz in eine andere, während das Alignment das *Ergebnis* dieser Mutation ist. Verschiedene Mutationen (=edit transcripts) können als Ergebnis das gleiche Alignment haben. *Deshalb betrachtet man das Alignment, da es vom Entstehungsprozess abstrahiert.*

2 Distanz von Sequenzen

Ähnliche Sequenzen von Baseresten haben ähnliche funktionelle Bedeutung. Man versucht nun in verschiedenen Sequenzen (z.B von verschiedenen Organismen) ähnliche Teile wiederzufinden, um dann die Funktion bestimmen zu können. Der Computer ist angehalten, möglichst gute Aligments zu finden, die den wirklichen Vorgängen entsprechen. Um die Qualität des Alignments in Zahlen zu fassen, benutzt man *Distanzen* oder *Ähnlichkeiten*. Damit läßt sich dann auch das beste Alignment finden. Um diese Distanzen oder Ähnlichkeiten zu berechnen führt man fiktive Kosten ein, die dann minimiert werden.

Zunächst beschäftige ich mit mit der Editierdistanz. Ein Sequenz wird mit den o.g. Editieroperationen in eine andere Sequenz überführt. Dabei können die verschiedenen Operationen verschiedene (fiktive) Kosten haben. Eine Kostenfunktion $d(a, b)$ definiert die Kosten für das Überführen eines einzelnen Zeichens, da die Wahrscheinlichkeit für das Ersetzen von verschiedenen Basenresten unterschiedlich ist.

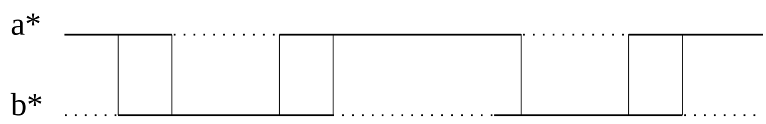
Definition Die Editierdistanz (oder kurz: „Distanz") zweier Sequenzen ist die Anzahl der Editieroperationen um die eine Sequenz in die andere zu überführen. Dies entspricht der Länge des edit transcript. Da es mehrere mögliche edit transscripts gibt, meint die Distanz eine kürzeste[1] Distanz.

Um ein optimales Alignment zu bekommen, müssen die Kosten, die entstehen, um eine ganze Sequenz zu überführen, minimiert werden. Dazu bedient man sich einer rekursiven Vorgehensweise. Man betrachtet einen optimalen Prefix der Editierfolge und wendet eine weitere Operation an, die wiederum eine optimale Folge als Ergebnis hat. Führt man das bis zum letzten Zeichen fort, und definiert man ein Anfangselement, dann erhält man eine optimale Editierfolge bzw. Alignment.

Seien $a = a_1 a_2 \cdots a_n$ und $b = b_1 b_2 \cdots b_m$ Sequenzen über dem endlichen Alphabet $\mathcal{A}$. Außerdem seien $d(a, b)$ die Kosten[2] um ein Zeichen a in ein Zeichen b zu überführen. Die Distanz stellt die Kosten für eine Veränderung (oder Mutation) der Sequenz von a nach b dar. Das Optimum aller Alignments (= minimale Distanz) von a nach b ist durch

$$D(a, b) = \min \sum_{i=1}^{L} d(a_i^*, b_i^*) \tag{2}$$

definiert. a^* und b^* sind die durch das Alignment veränderten Sequenzen, bei denen Null-Zeichen eingefügt wurden. Es wird minimiert über die Summe der Alignmentkosten aller möglichen Alignments. Beide Sequenzen a^* und b^* haben die gleiche Länge $(L = n = m)$:



Die Laufzeit um alle möglichen Kominationen durchzuprobieren ist $O\binom{n+m}{n}$.

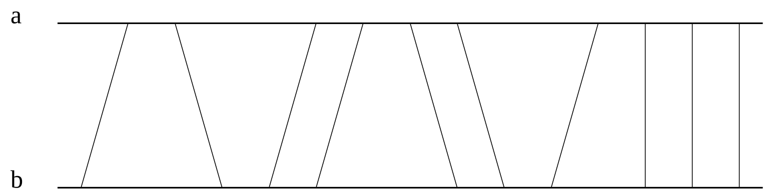
Im folgenden Kapitel wird dargestellt, wie man das Alignment (2) effektiv berechnen kann.

2.1 Globales Alignment

Das durch die Distanz (2) definierte Alignment nennt man *global*, da alle Zeichen der Sequenz a mit allen Zeichen der Sequenz b aligned werden:

[1]Die kürzeste Distanz muß nicht immer die minimale Anzahl von Operationen bedeuten. Es ist auch möglich verschiedene Editieroperationen unterschiedlich stark zu gewichten.

[2]Hier werden für verschiedene Zeichen unterschiedliche Kosten angesetzt. Zunächst kann man aber mit $d(a, b) = 1$, $d(a, a) = 0$ und $d(a, -) = d(-, b) = 1$ rechnen.



Man möchte jedoch nicht — wie in der Definition (2) — alle möglichen Alignments berechnen um das optimale Alignment zu erhalten. Deshalb wird jetzt eine Methode vorgestellt, die es erlaubt auf bekannten bzw. schon errechnetem Wissen aufzubauen und eine optimale Lösung zu erhalten. Dazu berechnet man die Distanzen rekursiv. Jeder Rekursionsschritt baut auf den davor berechneten Schritten auf.

Definition Mit zwei Sequenzen $a = a_1 a_2 \cdots a_n$ und $b = b_1 b_2 \cdots b_m$ definiere die Distanz-Matrix

$$D_{i,j} = D(a_1 a_2 \cdots a_i, b_1 b_2 \cdots b_j).$$

Dies ist die rekursive Definition der Distanz-Matrix. Ein Eintrag (i, j) in der Matrix entspricht der Distanz der Teilsequenzen $a_1 \cdots a_i$ und $b_1 \cdots b_j$. Man betrachtet also zunächst Teilsequenzen und zum Schluß erst das komplette Alignment $D_{n,m}$.

Außerdem setze

$$D_{00} = 0, D_{0,j} = \sum_{k=1}^{j} d(-, b_k), \text{ und } D_{i,0} = \sum_{k=1}^{i} d(a_k, -). \tag{3}$$

Dies ist der Rekursionsanfang. Die null-te Zeile und Spalte werden durch Anfangskosten belegt, von denen aus die Matrix berechnet werden kann. Die null-te Zeile entspricht dem Alignment

```
abc
---
```

Die null-te Spalte entspricht dem Alignment

```
---
def
```

Man spricht auch von einer „recurrence relation”, da die Distanz rekursiv definiert ist.

Nun betrachtet man optimale Prefixe einer Editierfolge. Hat man eine optimale Folge gegeben, kann man ein Zeichen (oder indel) anfügen, so daß die entstehende Folge wieder optimal ist. Dazu betrachtet man die Distanz der optimalen Folge und fügt ein Zeichen so an, daß die Gesamtdistanz wieder optimal ist.

Satz Ein Alignment $D_{i,j}$ basiert auf optimalen Prefixen bei denen eine der drei Editieroperationen angewendet wird. Die Distanz ist dann rekursiv zu berechnen durch

$$D_{i,j} = \min \left\{ \begin{array}{l} D_{i-1,j} + d(a_i, -), \\ D_{i-1,j-1} + d(a_i, b_j), \\ D_{i,j-1} + d(-, b_j) \end{array} \right\}. \tag{4}$$

Beweis Um zu zeigen, daß Formel (4) ein optimales Alignment berechnet, betrachtet man das Ende der Alignments von $a_1 a_2 \cdots a_i$ und $b_1 b_2 \cdots b_j$. Es kann nur auf eine von drei Arten enden:

$$\begin{array}{ccc} \cdots a_i & \cdots a_i & \cdots - \\ \cdots - & \cdots b_j & \cdots b_j \end{array}$$

Man betrachte eine Transformation der Sequenzen $a_1 \cdots a_i$ nach $b_1 \cdots b_j$ mit einer minimalen Anzahl von Editieroperationen. Für das letzte Zeichen der Editierfolge gibt es vier Möglichkeiten (I,D,R,M):

1. Ist das letzte Zeichen ein I, dann ist die letzte Operation das Anhängen eines Zeichens an die bis dahin transformierte Sequenz. Die Editierfolge vor dem I muß eine minimale Anzahl von Editieroperationen enthalten (wenn nicht, dann wäre die Editierfolge von $a_1 \cdots a_i$ nach $b_1 \cdots b_j$ nicht minimal). Laut Definition sind dafür $D_{i,j-1}$ Operationen notwendig. Für das Anhängen des Zeichens gilt dann $D_{i,j} = D_{i,j-1} + d(-, b_j)$.

2. Ist das letzte Zeichen ein D, dann ist die letzte Operation das Löschen des letzten Zeichens. Die Editierfolge vor dem D muß eine minimale Anzahl von Editieroperationen enthalten. Laut Definition sind dafür $D_{i-1,j}$ Operationen notwendig. Für das Anhängen des Zeichens gilt dann $D_{i,j} = D_{i-1,j} + d(a_i, -)$.

3. Ist das letzte Zeichen ein R, so wird das Zeichen a_i durch b_j ersetzt. Die Editierfolge vor dem R muß eine minimale Anzahl von Editieroperationen enthalten, um $a_1 \cdots a_{i-1}$ nach $b_1 \cdots b_{j-1}$ zu transformieren. In diesem Fall ist $D_{i,j} = D_{i-1,j-1} + d(a_i, b_j)$.

4. Sind die beiden letzten Zeichen gleich, so gilt $D_{i,j} = D_{i-1,j-1}$.

Die beiden letzten Fälle sind im Allgemeinen zusammengefaßt laut der Definition von

1. $d(a, b) = 0$ wenn $a = b$,

2. $d(a, b) = 1$ wenn $a \neq b$.

Die optimale Editiersequenz hat mindestens die Länge von einer dieser drei Möglichkeiten. Das optimale Alignment hat die damit verbundenen Kosten. Man beginnt laut vollständiger Induktion mit der null-ten Zeile und null-ten Spalte und schließt dann auf $D_{n,m}$. ■

3 Dynamische Programmierung

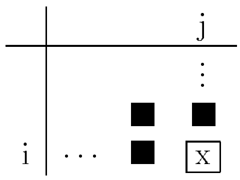
Dynamische Programmierung ist ein Verfahren, das in vielen Gebieten der Informatik angewendet wird. Die Hauptidee ist es, das Problem (hier minimale Editierdistanz) auf ein Teilproblem zurückzuführen, das einfach gelöst werden kann. Dabei werden im Gegensatz zu Divide-and-Conquer Teilergebnisse immer wieder verwendet.

Die Vorraussetzung für dynamische Programmierung ist zum einen das Vorhandensein einer optimalen Teilstruktur. Dies ist hier gegeben, da alle Prefixe eines optimalen Alignments selbst optimal sind. Zum anderen müssen sich die Teilprobleme überlappen. Auch das ist hier erfüllt. Prefixe von Sequenzen kommen in mehreren Sequenzen vor.

Die Bestandteile der dynamischen Programmierung sind immer (1) Rekursion, (2) Tabellenerstellung und (3) traceback. Bei dem Editierdistanzproblem wird genau so vorgegangen:

Rekursion Zunächst wird das Problem rekursiv definiert und in kleinere Stücke zerteilt. Man berechnet die Distanz $D_{i,j}$ für $1 \leq i \leq n$ und $1 \leq j \leq m$, um schließlich auf die Gesamtdistanz $D_{n,m}$ zu kommen.

Tabellenerstellung Dazu initialisiert man eine Distanztabelle (oder Matrix) mit der Formel (3). Dabei wird die erste Zeile und erste Spalte mit Werten gefüllt. Dann füllt man die restliche Tabelle laut Formel (4). Bei jedem Eintrag der Matrix muß nur der Eintrag links, oben und diagonal links-oben betrachtet werden:



traceback Man merkt sich bei jeder Position (i, j) einen oder mehrere Zeiger, welche der drei Möglichkeiten $(i-1, j)$, $(i-1, j-1)$ oder $(i, j-1)$ zu einer optimalen Lösung von $D_{i,j}$ geführt hat. Beginnend mit dem Element $D_{n,m}$ verfolgt man die Zeiger zurück und erhält einen optimalen Pfad. Sind bei einer Position mehrere Zeiger vermerkt, gab es also mehrere Möglichkeiten zu diesem Distanzwert zu kommen, dann bedeutet das, daß es mehrere mögliche Pfade gibt.

Laufzeit Man kann leicht sehen, daß die Laufzeit $O(nm)$ ist, da für das Eintragen der Werte in die Matrix nur die Zellen $(i-1, j)$, $(i, j-1)$ und $(i-1, j-1)$ betrachtet werden müssen. Die Matrix hat n mal m Einträge.

3.1 Beispiel

Ich möchte an einem Beispiel ,,vintner-writers" zeigen, wie man mit Hilfe von dynamischer Programmierung die Alignments berechnen kann. Ich setze der Einfachheit halber die Kosten

- $d(a, b) = 1$, wenn $a \neq b$ ist,
- $d(a, b) = 0$, wenn $a = b$ ist,
- $d(a, -) = d(-, a) = 1$.

Die initiale Matrix (Rekursionsanfang) sieht dann laut Formel (3) so aus:

$D_{i,j}$			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1							
i	2	2							
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

Geht man von der o.g. Tabelle aus und füllt die Einträge zeilen- oder spaltenweise, dann erhält man die folgende Matrix:

$D_{i,j}$			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4	*			
n	5	5							
e	6	6							
r	7	7							

An der Stelle des Sterns würde nun das nächste Element eingetragen. Es wäre eine 3, da das Zeichen t an in beiden Strings an der Stelle vorkommt.

Die komplette Tabelle inklusive der traceback-Zeiger sieht dann so aus:

$D_{i,j}$	0	w 1	r 2	i 3	t 4	e 5	r 6	s 7
0	0♣	$\leftarrow$ 1	$\leftarrow$ 2	$\leftarrow$ 3	$\leftarrow$ 4	$\leftarrow$ 5	$\leftarrow$ 6	$\leftarrow$ 7
v 1	$\uparrow$ 1	$\nwarrow$ 1♣	$\nwarrow\leftarrow$ 2	$\nwarrow\leftarrow$ 3	$\nwarrow\leftarrow$ 4	$\nwarrow\leftarrow$ 5	$\nwarrow\leftarrow$ 6	$\nwarrow\leftarrow$ 7
i 2	$\uparrow$ 2	$\nwarrow\uparrow$ 2	$\nwarrow$ 2♣	$\nwarrow$ 2	$\leftarrow$ 3	$\leftarrow$ 4	$\leftarrow$ 5	$\leftarrow$ 6
n 3	$\uparrow$ 3	$\nwarrow\uparrow$ 3	$\nwarrow\uparrow$ 3	$\nwarrow\uparrow$ 3♣	$\nwarrow$ 3	$\nwarrow\leftarrow$ 4	$\nwarrow\leftarrow$ 5	$\nwarrow\leftarrow$ 6
t 4	$\uparrow$ 4	$\nwarrow\uparrow$ 4	$\nwarrow\uparrow$ 4	$\nwarrow\uparrow$ 4	$\nwarrow$ 3♣	$\nwarrow\leftarrow$ 4	$\nwarrow\leftarrow$ 5	$\nwarrow\leftarrow$ 6
n 5	$\uparrow$ 5	$\nwarrow\uparrow$ 5	$\nwarrow\uparrow$ 5	$\nwarrow\uparrow$ 5	$\uparrow$ 4♣	$\nwarrow$ 4	$\nwarrow\leftarrow$ 5	$\nwarrow\leftarrow$ 6
e 6	$\uparrow$ 6	$\nwarrow\uparrow$ 6	$\nwarrow\uparrow$ 6	$\nwarrow\uparrow$ 6	$\uparrow$ 5	$\nwarrow$ 4♣	$\nwarrow\leftarrow$ 5	$\nwarrow\leftarrow$ 6
r 7	$\uparrow$ 7	$\nwarrow\uparrow$ 7	$\nwarrow$ 6	$\nwarrow\uparrow$ 7	$\uparrow$ 6	$\uparrow$ 5	$\nwarrow$ 4♣	$\leftarrow$ 5♣

Ein möglicher Pfad ist durch ♣ markiert. Das markierte Alignment ist

```
writ-ers
vintner-
```

Außerdem sind noch die optimalen Alignments

```
wri-t-ers
-vintner-
```

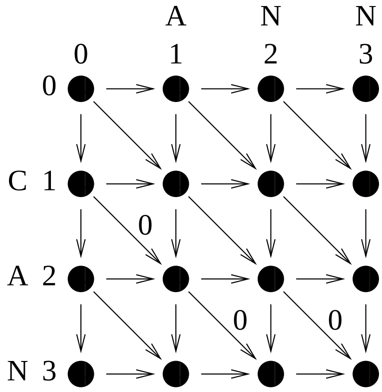
und

```
wri-t-ers
v-intner-
```

möglich.

3.2 Edit Graphs

Es ist auch möglich das Editiersequenz-Problem als Graph mit $(n+1) \times (m+1)$ Knoten darzustellen. Die gewichteten Kanten stellen die möglichen Editieroperationen dar. Es existiert eine gerichtete Kante von jedem Knoten (i, j) zu den Knoten $(i+1, j)$, $(i, j+1)$ und $(i+1, j+1)$ sofern diese existieren. Das Gewicht der ersten beiden Kanten ist 1; das Gewicht der Diagonale ist 0, wenn die Zeichen gleich sind und 1 wenn sie unterschiedlich sind.



Nun benutzt man einen *shortest path* Algorithmus (Pfad mit niedrigstem Gewicht) um einen Pfad von $(0,0)$ zu (n,m) zu finden. Dieser stellt eine optimale Editiersequenz dar.

Der Vorteil dieser Vorgehensweise liegt darin, daß man einen wohlbekannten Algorithmus verwenden kann um das Problem zu lösen.

4 Gaps

Bei der Evolution von Sequenzen kommt es oft vor, daß nicht einzelne Zeichen gelöscht oder eingefügt werden, sondern ganze Ketten von Zeichen. Diese aufeinanderfolgenden Operationen sind dann anders zu gewichten als einzelne Einfüge- und Lösch-Operationen.

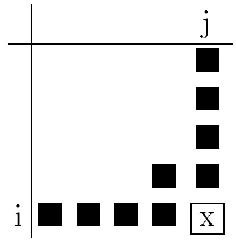
Man definiert eine Funktion $g(k)$, die die Kosten einer k-fachen indel-Operation darstellt. Diese k-fachen indel nennt man auch *gaps*. Es wird sinnvoller Weise angenommen, daß $g(k) \leq kg(1)$. Bei jedem Eintrag in der Distanzmatrix muß man nun nicht nur das einfache Einfügen und Löschen der benachbarten Zeichen betrachten, sondern auch das Löschen von mehreren Zeichen. Dazu muß man natürlich in der Distanzmatrix die darüberliegende Spalte und die danebenliegen Zeile betrachten.

Es ergibt sich eine neue Berechnungsvorschrift der Distanzmatrix.

Satz Sei $D_{i,j} = D(a_1 a_2 \cdots a_i, b_1 b_2 \cdots b_j)$, $D_{0,0} = 0$, $D_{0,j} = g(j)$ und $D_{i,0} = g(i)$. Dann läßt sich die Distanz berechnen durch

$$D_{i,j} = \min \left\{ \begin{array}{l} D_{i-1,j-1} + d(a_i, b_j), \\ \min_{1 \leq k \leq j} \{D_{i,j-k} + g(k)\}, \\ \min_{1 \leq l \leq i} \{D_{i-l,j} + g(l)\} \end{array} \right\} \tag{5}$$

Bei der Berechnung eines Eintrags der Matrix müssen nun nicht nur die drei benachbarten Felder, sondern auch die ganze darüberliegende Spalte und die ganze linke Zeile berechnet werden:



Beweis Der Beweis geht analog zum letzten mit der Erweiterung, daß ein Alignment auch mit z.B. i Löschungen der Sequenz $a_1 a_2 \cdots a_i$ enden kann. Deshalb müssen nun in der Rekursion auch die möglichen Löschungen mit unterschiedlichen Kosten beachtet werden.

Die Komplexität ist $O(n^2m + nm^2)$ oder $O(n^3)$ für $n = m$.

4.1 Gewichtete Gaps

Entscheidend für die Berechnung von Alignments mit Gaps sind die Kosten $g(k)$. Man betrachtet vier Fälle:

1. Bei **konstanten** Kosten ist jedes indel „kostenlos" und jedem Gap beliebiger Länge werden Kosten zugeordnet. Dies ist der einfachste Fall. (Laufzeit $O(nm)$)
2. Bei **affinen** oder **linearen** Kosten gibt es Anfangskosten für das Gap und Erweiterungskosten ($g(k) = a + b(k-1)$). Diese Art wird im nächsten Abschnitt genauer betrachtet. (Laufzeit $O(nm)$ mögl.)
3. Bei **konvexen/konkaven** Kosten werden wieder Anfangskosten gegeben. Jedes weitere indel bekommt ein stetig fallendes/steigendes Kostenmaß zugeordnet. (Laufzeit konvex $O(nm \log m)$ oder mehr)
4. Bei **willkürlichen** Kosten werden einfach jeder Gap-Länge Kosten zugeordnet, die unabhängig von den anderen Gap-Kosten sind. (Laufzeit $O(nm^2 + n^2m)$)

Gaps mit linearen Kosten

Besonders bei Sequenzen der Länge 1000 und mehr ist es interessant, die Rechenzeit von $O(n^3)$ zu reduzieren. Im einfachsten Fall setzt man für $g(k)$ eine lineare Funktion an. Das führt zu einer vereinfachten Berechnung der Distanz.

Wenn man die Kosten von $k-1$ indels bis zur aktuellen Stelle kennt, dann kann man die k-fachen indel-Kosten für die aktuellen Stelle berechnen, indem man einfach einen konstanten Wert addiert. Bei einem match setzt man die indel-Kosten auf einen Anfangswert.

Definition Sei $g(k) = \alpha + \beta(k-1)$ für konstante α und β eine solche lineare Kostenfunktion. Außerdem definiere man zwei weitere Matrizen E und F rekursiv durch

$$
\begin{aligned}
E_{0,0} &= F_{0,0} = D_{0,0} = 0, \\
E_{i,0} &= D_{i,0} = g(i), \\
F_{0,j} &= D_{0,j} = g(j), \\
E_{i,j} &= \min\{D_{i,j-1} + \alpha, E_{i,j-1} + \beta\} \quad \text{und} \\
F_{i,j} &= \min\{D_{i-1,j} + \alpha, F_{i-1,j} + \beta\}.
\end{aligned}
$$

Dann berechnet sich die Distanzmatrix durch

$$
D_{i,j} = \min \left\{ \begin{array}{l} D_{i-1,j-1} + d(a_i, b_j), \\ E_{i,j}, \\ F_{i,j} \end{array} \right\}.
$$

Hier wird in den Matrizen E und F die Kosten für das Löschen von mehreren Zeichen zwischengespeichert. Dies entspricht den Kosten der letzten beiden Zeilen in Formel (5), also den minimalen Kosten um $a_1 \cdots a_i$ nach $b_1 \cdots b_j$ zu überführen, wobei der letzte Schritt eine Löschoperation ist.

In E stehen die Kosten bis zum letzten match/mismatch (inkl.) plus den (geringeren) indel Kosten bis zur aktuellen Stelle in derselben Zeile. In F sind die Spaltenkosten abgelegt. So braucht man nicht in jedem Feld (i, j) die darüberliegenden Kosten erneut zu berechnen, sondern kann sie in den Matrizen E und F ablesen. Die gesamte Berechnung läuft wie folgt:

- Die Matrizen werden initialisiert. Dazu werden die erste Zeile und Spalte mit den Kosten $g(i)$ bzw. $g(j)$ gefüllt.
- Der Matrixeintrag $E_{i,j}$ wird berechnet. Das ist unabhängig von den noch nicht berechneten Einträgen $D_{i,j}$ und $F_{i,j}$, da nur auf Werte oberhalb und links zugegriffen wird.
- Der Matrixeintrag $F_{i,j}$ wird berechnet. Das ist unabhängig von den noch nicht berechneten Einträgen $D_{i,j}$ und $E_{i,j}$, da nur auf Werte oberhalb und links zugegriffen wird.
- Der Matrixeintrag $D_{i,j}$ wird berechnet. Das ist unabhängig von den noch nicht berechneten Einträgen $E_{i,j}$ und $F_{i,j}$, da nur auf Werte oberhalb und links zugegriffen wird.

Damit wird — wie gehabt — die Matrix zeilen- oder spaltenweise gefüllt. Die Pointer werden wieder mitgeschrieben, und im traceback liest man den optimalen Pfad aus.

Laufzeit Die Laufzeit wird durch dieses Verfahren wieder auf $O(nm)$ gedrückt. Bei jedem Eintrag in die Tabelle wird nur eine konstante Zahl von Operationen ausgeführt. Auch der Speicherplatzbedarf steigt nur konstant.

4.2 konvex / konkav

Die Funktion $g(k)$ heißt konkav[3], wenn

$$g(k+m+l) - g(k+m) \leq g(k+l) - g(k)$$

für alle $k, l, m \geq 0$. Das bedeutet, daß $g(k)$ so geartet ist, daß das Kostenmaß zwischen zwei Werten mit zunehmenden k abnimmt. Ein Beispiel ist z.B. $g(k) = \alpha + \beta \log(k)$. Laufzeiten von $O(n^2 \log(n))$ sind möglich.

Für konvexe Funktionen dreht sich das Ungleichheitszeichen um. Konvexe $g(k)$ erleichtern das Finden eines $O(nm)$ Algorithmus, jedoch ist diese Situation in der Biologie eher unwahrscheinlich.

5 Ähnlichkeit von Sequenzen

Alternativ zu Distanzen kann man auch ein Maß für die Ähnlichkeit von Sequenzen angeben. Distanzen beziehen sich in der Regel auf ganze Sequenzen, die auch als Ganzes aligned werden (global Alignment). Ähnlichkeitsmaße kann man aber auch auf Teilsequenzen, d.h. eine kurze Sequenz in einer langen, angeben.

Die Idee von Ähnlichkeiten beruht auf dem ,,Belohnung–Bestrafung" Prinzip. Zwei gleiche Zeichen (match; identity) in einem Alignment werden mit einem Ähnlichkeitswert $s(a,a) > 0$ belohnt. Zwei verschiedene Zeichen (mismatch) werden mit einem Wert $s(a,b) < 0$ bestraft. Hier werden wieder für verschiedene Zeichen unterschiedliche Bestrafungen s abgesetzt.

Außerdem ist $-\hat{g}(a)$ das Bestrafungsmaß[4] für das indel eines Zeichens a. Es ist definiert durch

$$-\hat{g}(a) = s(a, -) = s(-, a).$$

Die Ähnlichkeit (similarity) S von a und b ist dann definiert durch

$$S(a,b) = \max \sum_{i=1}^{L} s(a_i^*, b_i^*),$$

die das Ähnlichkeitsmaximum über alle möglichen Alignments darstellt. Für eine Sequenz wird wieder — wie bei der Distanz — rekursiv eine Matrix mit den Sequenzen $a = a_1 a_2 \cdots a_n$ und $b = b_1 b_2 \cdots b_m$ definiert:

$$S_{i,j} = S(a_1 a_2 \cdots a_i, b_1 b_2 \cdots b_j).$$

[3] Gelegentlich werden die beiden Begriffe in der Literatur vertauscht benutzt.

[4] Hier wird $\hat{g}$ benutzt, um von dem indel Gewicht bei Distanzen zu unterscheiden.

Dies ist die Ähnlichkeit der Teilsequenzen $a_1a_2\cdots a_i$ und $b_1b_2\cdots b_j$) auf einem bestimmten Pfad. $S_{m,n}$ ist dann die Ähnlichkeit der gesamten Sequenzen. Als Rekursionsanfang definiert man analog

$$S_{0,0}=0,\ S_{0,j}=\sum_{k=1}^{j}s(-,b_k),\ \text{und}\ S_{i,0}=\sum_{k=1}^{i}s(a_k,-).$$

Dann berechnet man jeden Eintrag der Matrix durch

$$S_{i,j}=\max\left\{\begin{array}{l} S_{i-1,j}+s(a_i,-),\\ S_{i-1,j-1}+s(a_i,b_j),\\ S_{i,j-1}+s(-,b_j)\end{array}\right\}.$$

Setzt man unabhängig von a alle $s(a,-)=s(-,a)=-\hat{\delta}$, dann vereinfacht sich die Formel zu

$$S_{i,j}=\max\left\{\begin{array}{l} S_{i-1,j}-\hat{\delta},\\ S_{i-1,j-1}+s(a_i,b_j),\\ S_{i,j-1}-\hat{\delta}\end{array}\right\}.$$

Die Berechnungen und Beweise gehen analog zur Distanz, nur daß hier über die Ähnlichkeitswerte maximiert und nicht über die Distanz minimiert wird. Außerdem lassen sich die Ähnlichkeiten auch auf kurze Folgen innerhalb von langen Folgen anwenden.

5.1 Äquivalenz von Ähnlichkeit und Distanz

In diesem Kapitel wird gezeigt, daß Ähnlichkeit und Distanz äquivalente Probleme sind. Wenn man zwei Sequenzen als Ganzes (global) mit einer Distanz (Ähnlichkeit) alignt, dann gibt es ein Ähnlichkeits- (Distanz-) Alignment, das die gleiche Menge von optimalen Alignments beinhaltet.

Satz Sei $s(a,b)$ ein Ähnlichkeitsmaß mit indel Strafen von $\hat{g}(k)$ und $d(a,b)$ eine Distanz mit dem indel Gewicht $g(k)$. Weiterhin gebe es eine Konstante c, so daß $s(a,b)=c-d(a,b)$ und $\hat{g}(k)=g(k)-(kc)/2$ gilt. Dann ist eine Alignment ähnlichkeits-optimal genau dann wenn es distanz-optimal ist.

Beweis Durch das einfache abzählen von Elementen erhählt man

$$n+m=2\#\text{matches}+\sum_k k\Delta_k,$$

wobei „#matches" die Anzahl der passenden Zeichen bedeutet und Δ_k die Anzahl der indels der Länge k ist. Durch diese einfache Form kann man die Distanz in eine Ähnlichkeit überführen:

$$D(a,b)\ =\ \min\left\{\sum_{\text{matches}}d(a,b)+\sum_k g(k)\Delta_k\right\}$$

$$
\begin{aligned}
&= \min\left\{\sum_{\text{matches}} c + \sum_k k\Delta_k c/2 - \sum_{\text{matches}} s(a,b) + \sum_k \hat{g}(k)\Delta_k\right\} \\
&= \min\left\{c(n+m)/2 - \sum_{\text{matches}} s(a,b) + \sum_k \hat{g}(k)\Delta_k\right\} \\
&= c(n+m)/2 - \max\left\{\sum_{\text{matches}} s(a,b) - \sum_k \hat{g}(k)\Delta_k\right\} \\
&= c(n+m)/2 - S(a,b)
\end{aligned}
$$

■

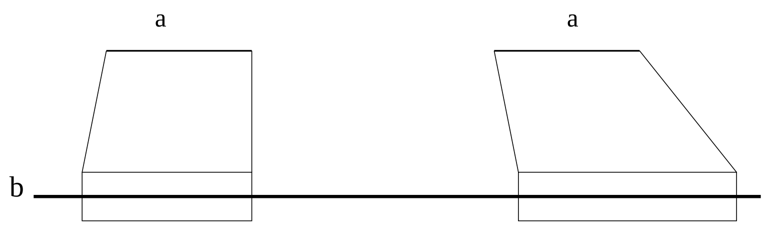
Normalerweise ist $0 \le c \le \max_{a',b'} d(a',b')$, so daß es positive und negative Ähnlichkeitswerte geben kann. Außerdem geht aus der Umformung

$$D(a,b) + S(a,b) = c(n+m)/2$$

hervor. So hängen Ähnlichkeitsmaß und Distanz direkt zusammen.

6 Alignment von kurzen Sequenzen

Als nächstes werden die Berechnungen so verändert, daß man eine kurze Sequenz in einer langen Sequenz einordnen kann:



Man betrachtet eine kurze Sequenz $a = a_1 a_2 \cdots a_n$, die in eine lange Sequenz $b = b_1 b_2 \cdots b_m$ eingebettet wird. Der Überschaubarkeit halber denke man sich n viel kleiner als m (das hat aber keine Auswirkung auf die Mathematik). Nun gilt es

$$T(a,b) = \max\{S(a, b_k b_{k+1} \cdots b_{l-1} b_l) : 1 \le k \le l \le m\}$$

zu finden. Das braucht die Laufzeit

$$\sum_{k=1}^{m} \sum_{k=l}^{m} n(l-k) = O(nm^3).$$

Um die Berechnung effizienter zu machen, wird der Algorithmus für globales Alignment wie folgt modifiziert: Es macht keinen Sinn, die „–” am Anfang und Ende von b mit Strafen zu belegen. Diese Löschungen liegen jeweils rechts und links vom Alignment. Die Matrix ist dann definiert durch

$$T_{i,j} = \max\{S(a_1a_2\cdots a_i, b_kb_{k+1}\cdots b_j) : 1 \le k \le j\}.$$

Für die Berechnung von $T_{i,j}$ setzt man als Rekursionsanfang $T_{0,j} = 0, 0 \le j \le m$ und $T_{i,0} = -i\delta$. Diese besondere erste Nullzeile verhindert, daß lange indels vor dem Alignment gewertet werden.

Dann berechnet man die Einträge der Matrix durch

$$T_{i,j} = \max\left\{\begin{array}{l} T_{i-1,j-1} + s(a_i, b_j), \\ T_{i,j-1} - \delta, \\ T_{i-1,j} - \delta \end{array}\right\}.$$

Die **Laufzeit** ist wieder auf $O(nm)$ reduziert, da nur die unmittelbaren Nachbarn betrachtet werden müssen.

Beispiel Die Sequenz a=TATAAT hat biologisch signifikate Bedeutung. Ich wende sie auf eine lange Sequenz (E. coli promoter of lacI) an mit den Werten $s(a,a) = 1$, $s(a,b) = -1$ und $\delta = 2$. Die Matrix sieht dann so aus:

	G	A	C	A	C	C	A	T	C	G	A	A	T
	0	0	0	0	0	0	0	0	0	0	0	0	0
T	−1	−1	−1	−1	−1	−1	−1	$\boxed{1}$	−1	−1	−1	−1	1
A	−3	0	−2	0	−2	−2	0	−1	$\boxed{0}$	−2	0	0	−1
T	−5	−2	−1	−2	−1	−3	−2	1	−1	$\boxed{-1}$	−2	−1	1
A	−7	−4	−3	0	−2	−2	−2	−1	0	−2	$\boxed{0}$	−1	−1
A	−9	−6	−5	−2	−1	−3	−1	−3	−2	−1	−1	$\boxed{1}$	−1
T	−11	−8	−7	−4	−3	−2	−3	0	−2	−3	−2	−1	$\boxed{2}$

	G	G	C	G	C	A	A	A	A	C	C	T	T
	0	0	0	0	0	0	0	0	0	0	0	0	0
T	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	1	1
A	0	−2	−2	−2	−2	0	0	0	0	−2	−2	−1	0
T	−1	−1	−3	−3	−3	−2	−1	−1	−1	−1	−3	−1	0
A	0	−2	−2	−4	−4	−2	−1	0	0	−2	−2	−3	−2
A	−2	−1	−3	−3	−5	−3	−1	0	1	−1	−3	−3	−4
T	0	−2	−2	−4	−4	−5	−3	−2	−1	0	−2	−2	−2

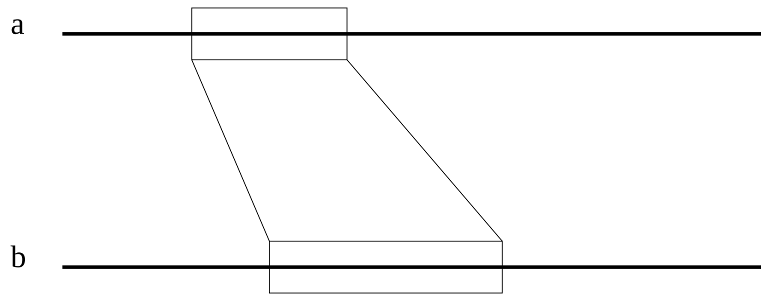
T	C	G	C	G	G	T	A	T	G	G	C	A	T	G	A	T
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	−1	−1	−1	−1	−1	1	−1	1	−1	−1	$\boxed{-1}$	−1	1	−1	−1	1
0	0	−2	−2	−2	−2	−1	2	0	0	−2	−2	$\boxed{0}$	−1	0	0	−1
1	−1	−1	−3	−3	−3	−1	0	3	1	−1	−3	−2	$\boxed{1}$	−1	−1	1
−1	0	−2	−2	−4	−4	−3	0	1	2	0	−2	−2	−1	$\boxed{0}$	0	−1
−3	−2	−1	−3	−3	−5	−5	−2	−1	0	1	−1	−1	−3	−2	$\boxed{1}$	−1
−3	−4	−3	−2	−4	−4	−4	−4	−1	−2	−1	0	−2	0	−2	−1	$\boxed{2}$

A	G	C	G	C	C	C	G	G	A	A	G	A	G	A	G	T
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	1
2	0	−2	−2	−2	−2	−2	−2	−2	0	0	−2	0	−2	0	−2	−1
0	1	−1	−3	−3	−3	−3	−3	−3	−2	−1	−1	−2	−1	−2	−1	−1
2	0	0	−2	−4	−4	−4	−4	−4	−2	−1	−2	0	−2	0	−2	−2
0	1	−1	−1	−3	−5	−5	−5	−5	−3	−1	−2	−1	−1	−1	−1	−3
0	−1	0	−2	−2	−4	−6	−6	−6	−5	−3	−2	−3	−2	−2	−2	0

Jeder Eintrag in der letzten Zeile markiert das Ende eines optimalen Pfades bis zu dieser Stelle. Man wählt die Maxima aus (hier: 2), da diese die besten Pfade markieren und damit auch die besten Alignments. Dann wendet man den traceback Algorithmus an und erhält zwei ähnliche Teil-Sequenzen.

7 lokales Alignment

Es kommt oft vor, daß zwei Ketten nicht sehr ähnlich sind. Jedoch gibt es Teilketten die in ähnlicher Form in beiden Ketten vorkommen. Das Finden solcher Teilketten nennt man *lokales Alignment*, wie in folgendem Schema:



Problemstellung Seien a und b zwei Sequenzen. Finde zwei Teilsequenzen α und β, deren Alignment optimal (maximale Ähnlichkeit) ist über alle möglichen Teilsequenzen aus a und b.

Am Beispiel der Sequenzen a=pqraxabcstvq und b=xyaxbacsll zeige ich die Idee. Ich setze als Kosten für ein match 2, für ein mismatch -2 und für ein Leerzeichen -1 ein. Dann ergibt sich für die beiden zufällig gewählten Teilsequenzen $\alpha = axabcs$ und $\beta = axbacs$ folgendes optimales (globales) Alignment mit den Kosten 8:

```
a x a b - c s
a x - b a c s
```

Das lokale Alignment fordert aber eine optimale Lösung über alle möglichen Teilstrings. Man benutzt das Ähnlichkeitsmaß anstelle der minimalen Distanz. Dies ist ersichtlich, da die minimale Distanz immer ein einzelnes passendes Zeichen ist und dadurch nicht cluster (oder Gruppen) gefunden werden können. Maximiert man die Ähnlichkeit von zwei Teilsequenzen, so findet man leichter ähnliche Regionen, da nebeneinander stehende matches höher bewertet werden.

Man nehme wieder eine Ähnlichkeit $s(a, b)$. Nun gilt es

$$H(a, b) = \max\{S(a_i a_{i+1} \cdots a_{j-1} a_j, b_k b_{k+1} \cdots b_{l-1} b_l) : 1 \leq i \leq j \leq n, 1 \leq k \leq l \leq m\}$$

zu finden. Das bedeutet, daß $\binom{n}{2}\binom{m}{2}$ Alignment Probleme gelöst werden müssen. Das wiederum führt zu einer Laufzeit $O(n^6)$, wenn $n = m$ ist.

7.1 Berechnung

Man kann jedoch das Problem wieder mit einer Matrix berechnen. Dabei werden dann (kurze) optimale Teilpfade gesucht, die dann das Alignment von Teilsequenzen darstellen:

$$\begin{array}{c|cccccc}
 & & k & & & l & \dots m \\ \hline
 & 0\dots & & & & & \dots 0 \\
 & & \vdots & & & & \\
i & \dots & \blacksquare & & & & \\
 & & & \blacksquare & & & \\
 & & & & \ddots & & \\
 & & & & \blacksquare & \vdots & \\
j & \dots & & & \dots & \blacksquare & \\
\vdots & & & & & & \\
n & \dots & & & & &
\end{array}$$

Die Idee ist es, zwei maximal ähnliche Sequenzen zu finden, die bei a_i und b_j enden:

$$H(a, b) = \max\{0; S(a_k a_{k+1} \cdots a_i, b_l b_{l+1} \cdots b_j) : 1 \leq k \leq i, 1 \leq l \leq j\}$$

Jeder Prefix einer Teilkette kann ignoriert werden. Dazu reinitialisiert die ,,0” die Rekursion.

Als Rekursionsanfang definiert man analog zu den oben angebenen Verfahren: $H_{i,0} = H_{0,j} = 0$ für alle $1 \leq i \leq n$ und $1 \leq j \leq m$. Die Ähnlichkeitsmatrix läßt sich berechnen durch:

$$H_{i,j} = \max \left\{ \begin{array}{l} 0, \\ H_{i-1,j-1} + s(a_i, b_j), \\ H_{i-1,j} + s(a_i, -), \\ H_{i,j-l} + s(-, b_j) \end{array} \right\},$$

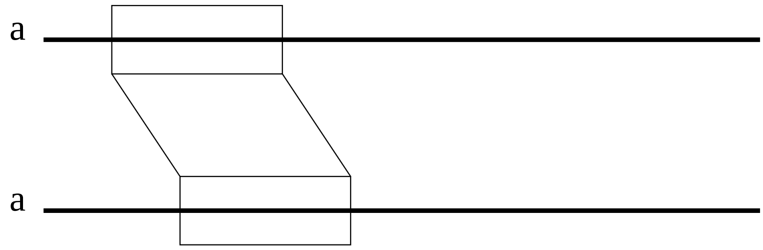
oder, wenn man Gaps in die Betrachtung mit einbezieht:

$$H_{i,j} = \max \left\{ \begin{array}{l} 0, \\ H_{i-1,j-1} + s(a_i, b_j), \\ \max_{1 \leq k \leq i}\{H_{i-k,j} - g(k)\}, \\ \max_{1 \leq l \leq j}\{H_{i,j-l} - g(l)\} \end{array} \right\}$$

Setzt man für die $g(k)$ lineare oder konstante Gewichte ein, so kann man wie oben mit den Zusatzmatrizen E und F Laufzeitverhalten von $O(nm)$ bekommen.

Um die Teilsequenzen zu erhalten, wendet man wieder traceback an. Dazu sucht man die Maxima in der gesamten Matrix (nicht nur letzte Zeile / letztes Feld). Von da aus verfolgt man die mitgeschriebenen Pointer rückwärts bis man auf eine 0 stößt. Diese markiert den Anfang der Teilsequenz, da hier die Rekursion (re-)initialisiert wurde.

Selbstähnlichkeiten Gelegentlich betrachtet man auch Selbstähnlichkeiten innerhalb einer Sequenz. Das Problem läßt jedoch auf das lokale Alignment von a mit sich selbst zurückführen.



Literatur

[1] Waterman, *Introduction to Computional Biology*, Chapman & Hall, 1995

[2] D. Gusfield, *Algorithms on Strings, Trees and Sequences*, CUP, 1997

[3] Cormen, Leiserson, Rivest, *Introduction to Algorithms*, MIT Press, 1990