

1 Sequenzalignment

- Motivation: Ähnlichkeit/Distanz von Wörtern
 - zum Beispiel: schimmlig/grimmig und Haus/Kaffee
 - Ähnlichkeit: schimmlig/grimmig
- Eigenschaften:
 - Mehrere gleiche Buchstaben
 - In der gleichen Reihenfolge
- Motivation 2: Homologe Proteinsequenzen

Zwei Proteine P,P' sind homolog, falls sie durch Evolution aus einem gemeinsamen Vorfahrprotein abstammen.

- Bemerkung Homologe Proteine haben meistens ähnliche Funktion und /oder Sequenzen
 - ⇒ Modell der Sequenzevolution wird benötigt.
- Evolution: Erlaubt Einfügen/Löschen und Ersetzen von Nukleotiden.
- Anzahl der notwendigen evolutionären Operationen ist ein Maß für die Distanz.

Definition 1.1 (Levenshtein Distanz für S_1, S_2) Minimale Anzahl von Einfügungen/Löschungen/Ersetzungen, die nötig sind um S_1 in zu S_2 überführen.

Beispiel 1.2 schimmlig $\xRightarrow{2\text{Loeschungen}}$ chimmig $\xRightarrow{2\text{Ersetzungen}}$ grimmig \Rightarrow Distanz 4
 Haus $\xRightarrow{3\text{Ersetzungen}}$ Kaff $\xRightarrow{2\text{Einfueugungen}}$ Kaffee \Rightarrow Distanz 8



Definition 1.3 (Edit Operation) Gegeben: Alphabet Σ (endliche Menge) mit $- \notin \Sigma$ (- gap-Symbol). Eine edit-Operation ist ein Paar $(x, y) \in (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\})$. (x, y) wird genannt:

- Substitution, falls $x \neq -$ und $y \neq -$
- Einfügung, falls $x = -$ und $y \neq -$
- Löschung, falls $x \neq -$ und $y = -$

Wir schreiben $a \xrightarrow{(x,y)} b$, falls b durch Ersetzen eines x durch y ((x,y) Substitution), bzw. eines Löschen von x ((x,y) Löschung), bzw. eines Einfügens eines y ((x,y) Einfügung) aus a generiert werden kann.

Beispiel 1.4 für Edit-Operationen:

- **Löschen** ATTACG $\rightarrow_{(T,-)}$ ATACG
- **Substitution** ACCA $\rightarrow_{(C,G)}$ ACGA
- **Einfügen** ATAT $\rightarrow_{(-,A)}$ ATAAT



Definition 1.5 Sei $S = o_1 \dots o_n$ eine Sequenz von Edit-Operationen. Wir schreiben $a \Rightarrow_S b$ gdw.

$$a = a^{(0)} \rightarrow_{o_1} a^{(1)} \rightarrow_{o_2} \dots \rightarrow_{o_n} a^{(n)} = b$$

Die erste und bekannteste Kostenfunktion in der Bioinformatik ist die Needleman-Wunsch-Kostenfunktion.

Definition 1.6 Die Kostenfunktion ist eine Funktion $w : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \rightarrow \mathbb{R}$.

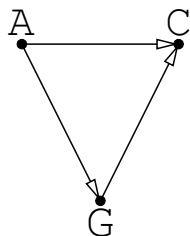
Die Kosten für eine Sequenz von Edit-Operationen $S = o_1 \dots o_n$ ist definiert als

$$w(S) = \sum_{i=1}^n w(o_i)$$

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

$$d_w(a, b) = \min\{w(S) \mid a \Rightarrow_S b\}$$

Problem: Es gibt unendlich viele Sequenzen, die a nach b überführen. Hierfür wird dynamisches Programmieren eingesetzt, um Teilprobleme nicht mehrfach zu lösen. Dafür benötigen wir eine Metrik. Die Grundidee ist, dass man für die Metrik keine Mehrfach-Ersetzungen lässt:



$$w(A,C) \leq w(A,G) + w(G,C)$$

Definition 1.7 Eine Kostenfunktion w heißt Metrik, falls sie folgende Bedingungen erfüllt:

1. $w(x, y) = 0 \leftrightarrow x = y$

- 2. $w(x, y) = w(y, x)$ (symmetrisch)
- 3. $w(x, z) \leq w(x, y) + w(y, z)$ (Dreiecksungleichung)

Definition 1.8 Gegeben sind 2 Wörter $a, b \in \Sigma^*$. 2 andere Wörter sind ein Alignment von (a, b) falls $a^*, b^* \in (\Sigma \cup \{-\})^*$ und:

- 1. $|a^*| = |b^*|$
- 2. $\forall_i \neg(a_i^* = - = b_i^*)$
- 3. $a^*|_{\Sigma} = a$ und $b^*|_{\Sigma} = b$

Beispiel 1.9 für ein mögliches Alignment:

| | | | | | | | |
|------|---|---|---|---|---|---|-----|
| a = | A | C | G | G | A | T | |
| b = | C | C | G | C | T | T | |
| a* = | A | C | - | G | G | - | A T |
| b* = | - | C | C | G | C | T | - T |



Definition 1.10 Die Kosten eines Alignments (a^*, b^*) bei gegebener Kostenfunktion w sind gegeben durch

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

Die Alignment-Distanz von (a, b) ist:

$$D(a, b) = \min\{w(a^*, b^*) \mid (a^*, b^*) \text{ Alignment von } (a, b)\}$$

Bemerkung 1.11 Falls w eine Metrik \rightarrow Edit-Distanz = Alignment-Distanz !

Lösung des Alignmentproblems: Dynamische Programmierung:

- 1. Problem läßt sich auf Teilprobleme zurückführen
- 2. Teilprobleme werden in Matrix gespeichert

Definition 1.12 Seien (a, b) zwei Wörter aus Σ^* . Die Matrix $(D_{ij})_{\substack{0 \leq i \leq |a| \\ 0 \leq j \leq |b|}}$ ist definiert durch:

$$D_{ij} = \min\{w(u^*, v^*) \mid (u^*, v^*) \text{ Alignment von } (a_1 \dots a_i, b_1 \dots b_j)\}$$

$\varepsilon \in \Sigma^*$ ist das leere Wort mit $|\varepsilon| = 0$.

Beispiel 1.13 $a = \text{AT}, b = \text{AAGT}$

$$w(x,y) = \begin{cases} 1 & \text{falls } x \neq y, \\ 0 & \text{sonst} \end{cases}$$

| | | | | | |
|---|---|---|---|---|--------|
| | | A | A | G | T |
| | 0 | 1 | 2 | 3 | 4 |
| A | 1 | 0 | | | |
| T | 2 | | | | D(a,b) |



Bemerkung 1.14 Sei (D_{ij}) Alignmentmatrix von a, b mit $|a| = n$ und $|b| = m$. Dann ist $D(a,b) = D_{n,m}$
Beispiel für Aufspaltung von Alignments

$$w \begin{pmatrix} A & C & - & G & G & - & A & T \\ - & C & C & G & C & T & - & T \end{pmatrix} = w \begin{pmatrix} A & C & - & G & G & - & A \\ - & C & C & G & C & T & - \end{pmatrix} + w \begin{pmatrix} T \\ T \end{pmatrix}$$

Proposition 1.15 Sei (u^*, v^*) ein Alignment von $(a_1 \dots a_i, b_1 \dots b_j)$. Sei $|u^*| = |v^*| = r$. Dann gilt:

$$w(u^*, v^*) = w(u_1^* \dots u_{r-1}^*, v_1^* \dots v_{r-1}^*) + w(u_r^*, v_r^*)$$

1. Löschen: Alignment von $(a_1 \dots a_{i-1}, b_1 \dots b_j)$

$$\dots \left| \begin{array}{l} a_i \\ - \end{array} \right. w(u^*, v^*) = D_{i-1,j} + w(u_r^*, v_r^*)$$

2. Einfügen: Alignment von $(a_1 \dots a_i, b_1 \dots b_{j-1})$

$$\dots \left| \begin{array}{l} - \\ b_j \end{array} \right. w(u^*, v^*) = D_{i,j-1} + w(u_r^*, v_r^*)$$

3. Ersetzen: Alignment von $(a_1 \dots a_{i-1}, b_1 \dots b_{j-1})$

$$\begin{array}{l|l} \dots & a_i \\ \dots & b_j \end{array} \quad w(u^*, v^*) = D_{i-1, j-1} + w(u_r^*, v_r^*)$$

Beispiel 1.16 für die Aufspaltung

$a = \text{AT}, i = 2, a_1 \dots a_2 = \text{AT}$

$b = \text{AAGT}, j = 3, b_1 \dots b_3 = \text{AAG}$

$$\begin{array}{l} \text{1. Fall: } \\ \text{2. Fall: } \\ \text{3. Fall: } \end{array} \begin{array}{l} u^* = \\ u^* = \\ u^* = \end{array} \begin{array}{l} - \\ A \\ - \end{array} \begin{array}{l} - \\ A \\ A \end{array} \begin{array}{l} A \\ G \\ A \end{array} \left| \begin{array}{l} T \\ - \\ T \\ G \\ A \\ G \end{array} \right.$$



Theorem 1.17 Sei $(D_{i,j})$ die Alignmentmatrix von (a,b) mit $|a| = n$ und $|b| = m$. Dann gilt:

$$D_{0,0} = 0$$

$$D_{i,0} = \sum_{k=1}^i w(a_k, -)$$

$$D_{0,j} = \sum_{k=1}^j w(-, b_k)$$

und für $1 \leq i \leq n, 1 \leq j \leq m$

$$D_{i,j} = \min \begin{cases} D_{i-1, j-1} + w(a_i, b_j) \\ D_{i-1, j} + w(a_i, -) \\ D_{i, j-1} + w(-, b_j) \end{cases}$$

Problem: Mit $D_{i,j}$ allein \Rightarrow nur Alignment Distanz, ist aber nicht automatisch bestes Alignment

1.1 Erweiterung: Tracematrix und Traceback

Idee: Für bestes Alignment merke für jedes $D_{i,j}$, wo kommt $D_{i,j}$ her-

Beispiel 1.18 Betrachte $D_{2,4}$ der Matrix

| | | | | | |
|---|---|---|---|---|---|
| | | A | A | G | T |
| | 0 | 1 | 2 | 3 | 4 |
| A | 1 | 0 | 1 | 2 | 3 |
| T | 2 | 1 | 1 | 2 | 2 |

Dann ist

$$D_{2,4} = \min \left\{ \begin{array}{l} D_{1,3} + w(T, T) \\ D_{2,3} + w(-, T) \\ D_{1,4} + w(T, -) \end{array} \right\} = D_{1,3} + w(T, T)$$

$$D_{1,3} = \min \left\{ \begin{array}{l} D_{0,2} + w(A, G) \\ D_{1,2} + w(-, G) \\ D_{0,3} + w(A, -) \end{array} \right\} = D_{1,2} + w(-, G)$$

⇒ Bestes Alignment:

- A - T
 A A G T



Definition 1.19 Die Tracematrix $(tr_{i,j})_{(0 \leq i \leq |a|) \text{ und } (0 \leq j \leq b)}$ für Alignment-matrix D und Wörter (a,b) ist eine Matrix mit Elementen $tr_{i,j} \subseteq \{\nwarrow, \uparrow, \leftarrow\}$ wobei folgendes gilt:

$$tr_{0,0} = \emptyset$$

$$tr_{0,j} = \{\leftarrow\}$$

$$tr_{i,0} = \{\uparrow\}$$

$$\forall_{i,j} \geq 1$$

$$\nwarrow \text{etr}_{i,j} \text{ genau dann wenn, } D_{i,j} = D_{i-1,j-1} + w(a_i, b_i)$$

$$\uparrow \text{etr}_{i,j} \text{ genau dann wenn, } D_{i,j} = D_{i-1,j} + w(a_i, -)$$

$$\leftarrow \text{etr}_{i,j} \text{ genau dann wenn, } D_{i,j} = D_{i,j-1} + w(-, b_j)$$

Beispiel 1.20 a=AT b=AAGT

| | | | | | |
|---|-----|-----|--------|--------|-----|
| | | A | A | G | T |
| | {∅} | {←} | {←} | {←} | {←} |
| A | {↑} | {↖} | {↖, ←} | {←} | {←} |
| T | {↑} | {↑} | {↖} | {←, ↖} | {↖} |

Pfad durch $tr_{i,j}$, von rechts unten bis links oben. Notiert von rechts nach links.

$$t_1 : \leftarrow \swarrow \leftarrow \swarrow$$

$$t_2 : \swarrow \leftarrow \leftarrow \swarrow$$



Definition 1.21 Sei t ein Traceback für (a,b) . Das dazugehörige Alignment ist dann wie folgt definiert.

$a^* =$ Ersetze i -tes Vorkommen von \swarrow, \uparrow durch a_i , jedes \leftarrow durch $-(gap)$

$b^* =$ Ersetze j -tes Vorkommen von \swarrow, \leftarrow durch b_j , jedes \uparrow durch gap

Beispiel 1.22 continued:

für t_1 : für t_2 :

$$a_1^* = -A - T \quad a_2^* = A - -T$$

$$b_1^* = AAGT \quad b_2^* = AAGT$$

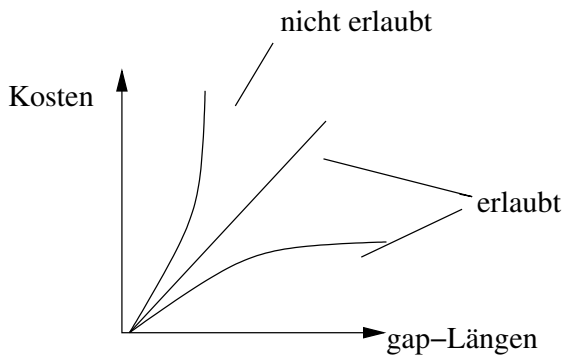


Verbleibendes Problem:

- evolutionäres Modell nicht korrekt
- Einfügen/Löschen: In der Wirklichkeit ist das Einfügen eines Gaps der Länge k leichter als k -mal Einfügen eines Gaps der Länge 1.

Idee: Das Starten eines Gaps ist teuer, das Verlängern billig.

Definition 1.23 Eine Gap-Penalty ist eine Funktion $g(k):N \Rightarrow R$ mit der Eigenschaft, dass g subadditiv ist: $g(k+l) \leq g(k) + g(l)$
 g heisst affin falls: $g(k) = \alpha + k\beta$ wobei α die Kosten für Gap-Einfügen sind und β die Kosten für Verlängerung.



1.2 Algorithmus für Alignment mit Gap-Penalties

Betrachte zunaechst die Fallunterscheidung des NEEDLEMAN/WUNSCH-Algorithmus. Hier kann man sehen, welche Faelle weiter behandelt werden muessen.

Beispiel 1.24 $a = ACGAT$
 $b = AGCAT$

$D_{2,3} \dots$ Kosten für bestes Alignment von AC und AGC

$$D_{2,3} = \min \begin{cases} D_{1,2} + w(C, C) \\ D_{2,2} + w(-, C) \\ D_{1,3} + w(C, -) \end{cases}$$

- ↔ mögliches Alignment:
- in letzter Spalte C,C
- in letzter Spalte -,C
- in letzter Spalte C, -



↔ Fallunterscheidung bei Rekursion \equiv Fallunterscheidung der letzten Spalte des besten Alignments

1. letzte Spalte = Substitution

... | a_i
 ... | b_j

Kosten: $D_{i-1,j-1}; w(a_i, b_j)$

↔ Gesamtkosten = $D_{i-1,j-1} + w(a_i, b_j)$

2. letzte Spalte = Insertion (Deletion analog)

$$\begin{array}{c} \dots \\ \dots \end{array} \left| \begin{array}{c} - \\ \bar{b}_j \end{array} \right.$$

Kosten: $D_{i,j-1}$; lokal $g(1)$ (für Gap)

aber: Anzahl der Gaps zu beachten

↔ Fallunterscheidung

(a) $\begin{array}{c} \dots \\ \dots \end{array} \left| \begin{array}{c} a_i \\ ? \end{array} \right| \begin{array}{c} - \\ \bar{b}_j \end{array} \Rightarrow \text{Gesamtkosten} = D_{i,j-1} + g(1)$

(b) $\begin{array}{c} \dots \\ \dots \end{array} \left| \begin{array}{c} - \\ b_{j-1} \end{array} \right| \begin{array}{c} - \\ \bar{b}_j \end{array} \Rightarrow \text{Gesamtkosten noch nicht berechenbar}$
 $\Rightarrow \text{weitere Fallunterscheidung...}$

- Rekursionsgleichung (WATERMAN/SMITH/BEYER)

$$D_{i,j} = \min \left\{ \begin{array}{l} D_{i-1,j-1} + w(a_i, b_j) \\ \min \left\{ \begin{array}{l} D_{i,j-1} + g(1) \\ D_{i,j-2} + g(2) \\ \vdots \\ D_{i,0} + g(j) \end{array} \right. \\ \min \left\{ \begin{array}{l} D_{i-1,j} + g(1) \\ D_{i-2,j} + g(2) \\ \vdots \\ D_{0,j} + g(i) \end{array} \right. \end{array} \right.$$

- kompaktere Formulierung

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

- Initialisierung:

- $D_{0,0} = 0$
- $D_{i,0} = g(i)$

$$- D_{0,j} = g(j)$$

- Berechnung:

- Initialisierung

- Tracematrix

- * für Alignment notwendig

- * Eintrag für Gap-Penalty und Substitution

- * $tr_{i,j} \subseteq \{\nwarrow, \leftarrow_k, \uparrow_l \mid k \leq j, l \leq i\}$

- * Traceback wie üblich (von rechts unten nach links oben, den Pfeilen folgend)

- * Erinnerung:

Def. von $tr_{i,j}$

$$\nwarrow \in tr_{i,j} \leftrightarrow D_{i,j} = D_{i-1,j-1} + w(a_i, b_j)$$

$$\uparrow_l \in tr_{i,j} \leftrightarrow D_{i,j} = D_{i-l,j} + g(l)$$

$$\leftarrow_k \in tr_{i,j} \leftrightarrow D_{i,j} = D_{i,j-k} + g(k)$$

- Komplexität:

- Annahme: $|a| = |b| = n$

- zu füllen: n^2 Zellen $\rightarrow O(n^2)$

- Zeitaufwand für das Füllen einer Zelle nicht konstant $\rightarrow O(n)$

\hookrightarrow Komplexität = $O(n^3)$

- Beispiel: durchschnittlicher Zeitaufwand für Zeile 1

Spalte 1: 2 (oben & diagonal) + 1 (links)

Spalte 2: 2 + 2

⋮

Spalte n: 2 + n

Summe: $2n + \frac{n(n+1)}{2}$

\hookrightarrow durchschnittliche Kosten für Elemente der 1. Zeile

$$\frac{2n + \frac{n(n+1)}{2}}{n} = 2 + \frac{n+1}{2} \equiv O(n)$$

- Beispiel:

- * 2 RNA-Sequenzen mit $n = 30000 = 3 \cdot 10^4$

- * $(3 \cdot 10^4)^3 = 27 \cdot 10^{12}$
- * Rechenzeit für einen Rechner mit 1 Ghz und 1 Operation pro Takt $\rightarrow 27000s \approx 7,5h$

• Verbesserung:

- Problem: Gap-Penalties
- Lösung: GOTOH's Algorithmus mit affinen Gap-Penalties

$$g(k) = \alpha + k \cdot \beta$$

- Betrachtung von $D_{i,j}$
- Fallunterscheidung nach letzter Spalte

$$1. \quad \begin{array}{c|c} \dots & a_i \\ \dots & b_j \end{array}$$

$$2. \quad \begin{array}{c|c} \dots & a_i \\ \dots & - \end{array}$$

$$(a) \quad \begin{array}{c|c} \dots & ? \\ \dots & b_j \end{array} \left| \begin{array}{c} a_i \\ - \end{array} \right. \text{Kosten} = D_{i-1,j} + g(1)$$

$$(b) \quad \begin{array}{c|c} \dots & a_{i-1} \\ \dots & - \end{array} \left| \begin{array}{c} a_i \\ - \end{array} \right. \begin{array}{l} \text{Kosten} = * + g(k) - g(k-1) \\ = * + \alpha + k \cdot \beta - (\alpha + (k-1) \cdot \beta) \\ = * + \beta \end{array}$$

*... Kosten für bestes Alignment von $a_1 \dots a_{i-1}$ mit $b_1 \dots b_j$, das mit Gap in b endet

\hookrightarrow Hilfsmatrizen

- * P_{ij} ... Kosten für bestes Alignment von $a_1 \dots a_i$ mit $b_1 \dots b_j$, das mit einem Gap in b endet
- * Q_{ij} ... Kosten für bestes Alignment, das mit einem Gap in a endet
- * Rekursionsgleichungen:

$$D_{ij} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) \\ P_{ij} \\ Q_{ij} \end{cases}$$

$$P_{ij} = \min \begin{cases} D_{i-1,j} + g(1) \\ P_{i-1,j} + \beta \end{cases}$$

$$Q_{ij} = \min \begin{cases} D_{i,j-1} + g(1) \\ Q_{i,j-1} + \beta \end{cases}$$

- Berechnung:

1. Initialisierung

2.


```

      FOR i=1 ... |a|
        FOR j=1 ... |b|
          Berechne P(i,j);
          Berechne Q(i,j);
          Berechne D(i,j);
      
```

- Komplexität: $O(n^2)$

- n-faches Zurückgreifen bei Berechnung von D_{ij} entfällt
- Komplexität von $P_{ij}, Q_{ij} = 4 = \text{konstant}$
- RNA-Sequenz mit $n = 30000$
 $\hookrightarrow (3 \cdot 10^4)^2 = 9 \cdot 10^8$
 Gigahetz-Prozessor benötigt ≤ 1 Sekunde

- Initialisierung:

- $D_{0,0} = 0; D_{0,j} = g(j); D_{i,0} = g(i);$
- Welche P_{ij}, Q_{ij} müssen initialisiert werden? $\rightarrow P_{0,j} \& Q_{i,0}$
- $P_{0,j} \dots$ Kosten für bestes Alignment mit $b_1 \dots b_j$, das mit einem Gap in b endet

\hookrightarrow $\begin{array}{cccccc} _ & _ & _ & _ & _ \\ b_1 & b_2 & \dots & b_j & _ \end{array}$ Widerspruch (schlechtes Alignment)

$\hookrightarrow P_{0,j} = \infty; Q_{i,0} = \infty$

- Tracematrix

- Ansatz: Füllen der Tracematrix wie bei NEEDLEMAN/WUNSCH

$$tr_{ij} \subseteq \{\nwarrow, \uparrow, \leftarrow\} \text{ mit}$$

$$\nwarrow \in tr_{ij} \leftrightarrow D_{ij} = D_{i-1,j-1} + w(a_i, b_j)$$

$$\uparrow \in tr_{ij} \leftrightarrow \text{ein Gap in b } (D_{ij} = P_{ij})$$

$$\leftarrow \in tr_{ij} \leftrightarrow \text{ein Gap in a } (D_{ij} = Q_{ij})$$

- Problem: falscher Ansatz, da 3 Kostenmatrizen
- Lösung: nicht 1 sondern 3 Tracematrizen mit für D, P, Q getrennten Elementen:

$$\{\nwarrow^D, \bullet^P, \bullet^Q, \uparrow^P, \uparrow^D, \leftarrow^Q, \leftarrow^D\}$$
- neue Definition der Tracematrix:

$$\forall_{ij} > 0 : \quad \begin{aligned} \nwarrow^D \in tr_{ij}^D &\leftrightarrow D_{ij} = D_{i-1,j-1} + w(a_i, b_j) \\ \bullet^Q \in tr_{ij}^D &\leftrightarrow D_{ij} = Q_{ij} \\ \bullet^P \in tr_{ij}^D &\leftrightarrow D_{ij} = P_{ij} \end{aligned}$$

$$\forall_{ij} > 0 : \quad \begin{aligned} \leftarrow^D \in tr_{ij}^Q &\leftrightarrow Q_{ij} = D_{i,j-1} + g(1) \\ \leftarrow^Q \in tr_{ij}^Q &\leftrightarrow Q_{ij} = Q_{i,j-1} + \beta \end{aligned}$$

$$\forall_{ij} > 0 : \quad \begin{aligned} \uparrow^D \in tr_{ij}^P &\leftrightarrow P_{ij} = D_{i-1,j} + g(1) \\ \uparrow^P \in tr_{ij}^P &\leftrightarrow P_{ij} = P_{i-1,j} + \beta \end{aligned}$$

• Beispiel GOTOH-Algorithmus

- a = C C
- b = A C C T
- g(k) = 4 + k
- $w(x.y) = \begin{cases} 1 & \text{falls } x \neq y \\ 0 & \text{sonst} \end{cases}$

$$- D_{ij} = \begin{array}{c|cccc} & & A & C & C & T \\ \hline & 0 & 5 & 6 & 7 & 8 \\ C & 5 & 1 & 5 & 6 & 8 \\ C & 6 & 6 & 1 & 5 & 7 \end{array}$$

$$P_{ij} = \begin{array}{c|cccc} & & A & C & C & T \\ \hline & \infty & \infty & \infty & \infty & \infty \\ C & ? & 10 & 11 & 12 & 13 \\ C & ? & 6 & 10 & 11 & 13 \end{array}$$

$$Q_{ij} = \begin{array}{c|cccc} & A & C & C & T \\ \hline C & \infty & 10 & 6 & 7 & 8 \\ C & \infty & 11 & 11 & 6 & 7 \end{array}$$

- Traceback: $\leftarrow \nearrow \nwarrow \leftarrow$

↪ Alignment: $\overline{} \quad \overline{C} \quad \overline{C}$
 $\phantom{\overline{}} \quad \overline{A} \quad \overline{C} \quad \overline{C} \quad \overline{T}$

Kosten = (4+1) + 0 + 0 + (4+1) = 10 → Widerspruch zu $D_{2,4} = 7$

- eigentliches Traceback:

| | | | | | |
|-------|---|----|----|----|----|
| | | A | C | C | T |
| | 0 | 5 | 6 | 7 | 8 |
| C | 5 | 1 | 5 | 6 | 8 |
| C | 6 | 6 | 1 | 5 | 7 |
| <hr/> | | | | | |
| | ∞ | - | - | - | - |
| C | ∞ | 10 | 6 | 7 | 8 |
| C | ∞ | 11 | 11 | 6 | 7 |
| <hr/> | | | | | |
| | ∞ | ∞ | ∞ | ∞ | ∞ |
| C | - | 10 | 11 | 12 | 13 |
| C | - | 6 | 10 | 11 | 13 |

1.3 Needleman-Wunsch-Algorithmus mit Ähnlichkeit

- bisher:
 - minimale Distanz gesucht
 - Kostenfunktion $w(x, x) = 0$ (Kosten für gleiche Zeichen gering)
 - Matrix D
- jetzt:
 - maximale Ähnlichkeit gesucht
 - Ähnlichkeitsfunktion $s(x, y)$, wobei Ähnlichkeit gleicher Zeichen hoch
 - Matrix S mit

$S_{i,j}$... beste Ähnlichkeit der Prefixe $a_1...a_i$ und $b_1...b_j$

– Rekursionsgleichung: $S_{i,j} = \max \begin{cases} S_{i-1,j} + s(a_i, -) \\ S_{i,j-1} + s(-, b_j) \\ S_{i-1,j-1} + s(a_i, b_j) \end{cases}$

- Haupteinsatz:

lokales Alignieren = Suche nach lokal ähnlichen Motiven

- Idee: Man möchte nicht vollständige Sequenzen sondern nur noch Teile alignieren.

Beispiel: CC mit ACCT

→ perfekter lokaler Match bzw. Domäne

- Was muss der Algorithmus leisten?
 - Finde Bereiche mit grösster lokaler Übereinstimmung (konservierte Domäne)
 - initiale und terminale Gaps bewertet man mit 0 (keine Kosten)

- Warum ist die Distanz hier nicht sinnvoll?

Bsp.:

$$\text{Dist.: } w(x, y) = \begin{cases} 0 & , x = y \\ 5 & , x \neq y \end{cases} \quad \text{Ähnl.: } s(x, y) = \begin{cases} 5 & , x = y \\ 0 & , x \neq y \end{cases}$$

1. X X A A C I X X
Y Y A A Y Y G

$$\rightarrow D \begin{pmatrix} AA \\ AA \end{pmatrix} = 0 \quad \text{und} \quad S \begin{pmatrix} AA \\ AA \end{pmatrix} = 10$$

2. X X A A A A X X
Y Y A A A A Y Y

$$\rightarrow D \begin{pmatrix} AAAA \\ AAAA \end{pmatrix} = 0 \quad \text{und} \quad S \begin{pmatrix} AAAA \\ AAAA \end{pmatrix} = 20$$

⇒ bezüglich der Distanz sind 1. und 2. gleich bewertet, aber 2. ist ein besseres lokales Motiv

→ das wird nur durch Ähnlichkeit sichtbar

Eine Ähnlichkeitsfunktion aus $\sigma \times \sigma$ ist eine Funktion $s(x, y)$
 $\sigma \times \sigma \rightarrow \mathbb{R}$ mit der Eigenschaft, dass $s(x, x)$ positiv ist.

Bem.: 1. Negative Werte definieren Unähnlichkeit.

Erinnerung: Pam-Matrix ist eine Ähnlichkeitsmatrix

2. Gaps werden häufig durch affine oder lineare Gap-Penalties festgelegt:

affine: $g(k) = \alpha + \beta k$

lineare: $g(k) = \delta k$

lineare gap penalty: Nach Definition: $g(k+1) = g(k) + g(1) \Rightarrow g(0) = 0$

⇒ $s(_, x) = s(x, _) = \delta$

Ähnlichkeit kann auf gap-Penalties erweitert werden.

3. Bei Ähnlichkeit haben gap-Penalties negative Werte!