

Vorlesung

# Effiziente Algorithmen und Komplexitätstheorie

Sommersemester 2008

Ingo Wegener

# Sequenzanalyse

Hauptproblem der Bioinformatik ist die computergestützte Sequenzanalyse, typischerweise auf

- $\Sigma = \{A, C, G, T\}$  (für die vier Basen Adenin, Guanin, Cytosin und Thymin)
- $\Sigma = \{A, C, G, U\}$  (mit Uracil statt Thymin)
- $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$  (Aminosäuren)

Dabei werden häufig zwei Sequenzen miteinander verglichen unter den Fragestellungen

- 1 Welche Art von Sequenzvergleich wollen wir vornehmen?
- 2 Wie bewerten wir die Ähnlichkeit zweier Sequenzen?
- 3 Welcher Algorithmus hilft dabei?
- 4 Wie können wir das Ergebnis auf Signifikanz überprüfen?

# String Matching

Eine Methode zur Sequenzanalyse haben wir bereits behandelt

## Definition

Die Suche nach allen Vorkommen eines Muster  $P \in \Sigma^*$  in einem Text  $T \in \Sigma^*$  (endliches Alphabet  $\Sigma$ ,  $n = |T|$ ,  $m = |P|$ ) heißt **String Matching**

## Theorem 6.18

Der Algorithmus von Knuth, Morris und Pratt löst das String-Matching-Problem in Zeit  $\Theta(n + m)$

## Theorem 6.28

Der Algorithmus von Boyer und Moore löst das String-Matching-Problem in Zeit  $O(|\Sigma| + m + n)$

Jetzt: Ähnlichkeit von Sequenzen

# Alignments mit Lücken

## Ähnlichkeit mittels Insertion und Delete

M o n **k** e y

M o n - e y

## Ähnlichkeit durch Substitution

**M** o n e y

**H** o n e y

Beobachtungen:

- Biologische Motivation: Mutation und Selektion
- Wir benötigen ein zusätzliches Symbol “-” (*Leerzeichen*)
- Die Ähnlichkeit zweier Sequenzen muss bewertet werden können

# Kostenfunktion von Alignments

## Definition

Eine Kostenfunktion (*score*)  $s : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \rightarrow \mathbb{R}$  für ein Ähnlichkeitsmaß heißt *sinnvoll*, wenn die folgenden Bedingungen erfüllt sind:

- 1  $\forall a \in \Sigma : s(a, a) \geq 0 \wedge s(a, -) \leq 0$
- 2  $\forall a, b \in \Sigma : s(a, a) \geq s(a, b)$
- 3  $\forall a, b \in \Sigma \cup \{-\} : s(a, b) = s(b, a)$
- 4  $\forall a, b \in \Sigma : s(a, b) \geq s(a, -) + s(-, b)$

# Alignments

## Definition

- 1 Ein *paarweises Alignment* ist ein Paar  $(x^*, y^*) \in (\Sigma \cup \{-\})^* \times (\Sigma \cup \{-\})^*$  mit  $k := |x^*| = |y^*|$  und  $x_i^* \neq - \neq y_i^*$  für alle  $1 \leq i \leq k$  mit  $x_i = y_i$ .
- 2  $(x^*, y^*)$  ist ein *globales Alignment* für  $x, y \in \Sigma^*$ , wenn  $x^*|_{\Sigma} = x$  und  $y^*|_{\Sigma} = y$  gilt.
- 3 Der *Wert* des Alignments gemäß Kostenfunktion  $s$  ist

$$s(x^*, y^*) := \sum_{1 \leq i \leq k} s(x_i^*, y_i^*) .$$

## Lösung mittels dynamischer Programmierung

Es sei  $S(i, j)$  die optimale Bewertung eines Alignments zwischen  $(x_1, \dots, x_i)$  und  $(y_1, \dots, y_j)$  und  $|x| =: n$ ,  $|y| =: m$

Weiter sei  $\forall a, b \in \Sigma : s(a, -) = s(-, b) = -d$  mit  $d > 0$

**Randwerte:**  $S(i, 0) = -id$  und  $S(0, j) = -jd$  für ein Alignment mit  $i$  bzw.  $j$  Leerzeichen

$$\begin{array}{cccc} x_1 & x_2 & \dots & x_i \\ - & - & \dots & - \end{array}$$

Möglichkeiten für  $S(i, j)$ :

$$\begin{array}{ccc} x_i & x_i & - \\ y_j & - & y_j \end{array}$$

Enthaltene Teilprobleme:  $(i - 1, j - 1)$ ,  $(i - 1, j)$  und  $(i, j - 1)$

# Algorithmus von Needleman und Wunsch

## Bellmansche Optimalitätsgleichung

$$S(i, j) = \max\{S(i-1, j-1) + s(x_i, y_j), S(i-1, j) - d, S(i, j-1) - d\}$$

## Theorem

*Der Wert eines optimalen globalen Alignments kann in Zeit  $O(|x| |y|)$  auf Platz  $O(\min\{|x|, |y|\})$  berechnet werden.*

**Beweis:** klar ✓

Die Aussage gilt für den **Wert** des Alignments

**Was ist mit dem Alignment selbst?**

# Berechnung des Alignments

Es ist leicht möglich das Alignment selbst zu berechnen.

Allerdings brauchen wir dafür die komplette Tabelle

Größe:  $(|x| + 1)(|y| + 1)$

## Idee:

- Starte “rechts unten” bei  $S(n, m)$  und führe Rückwärtssuche aus
- Speichere Zeiger gemäß den Fällen der Bellmannschen Optimalitätsgleichung

# Berechnung des Alignments

## Algorithmus

- ①  $i = n, j = m, z = 1$
- ② Solange  $i > 0$  und  $j > 0$ , unterscheide drei Fälle:
  - ①  $S(i, j) = S(i - 1, j - 1)$ :
    - ①  $x_z^* := x_i$  und  $y_z^* := y_j$
    - ②  $j := j - 1, i := i - 1, z := z + 1$
  - ②  $S(i, j) = S(i - 1, j) - d$ :
    - ①  $x_z^* := x_i$  und  $y_z^* := -$
    - ②  $i := i - 1, z := z + 1$
  - ③  $S(i, j) = S(i, j - 1) - d$ :
    - ①  $x_z^* := -$  und  $y_z^* := y_j$
    - ②  $j := j - 1, z := z + 1$
- ③ Reversiere  $x^*$  und  $y^*$

## Beispiel

 $x = A G C A, y = A T A$ 

Bewertung $s$	-	A	C	G	T
-	$-\infty$	-2	-1	-2	-1
A	-2	+1	-1	-2	-2
C	-1	-1	+1	-1	-2
G	-2	-2	-1	+1	-1
T	-1	-2	-2	-1	+1

$S$	0( $\emptyset$ )	1(A)	2(T)	3(A)
0( $\emptyset$ )	0	$\leftarrow -2$ -/A	$\leftarrow -3$ --/AT	$\leftarrow -5$ ---/ATA
1(A)	$\uparrow -2$ A/-	$\nearrow +1$ A/A	$\leftarrow 0$ A-/AT	$\leftarrow -2$ A--/ATA
2(G)	$\uparrow -4$ AG/--	$\uparrow -1$ AG/A-	$\nearrow 0$ AG/AT	$\nearrow -2$ A-G/ATA
3(C)	$\uparrow -5$ AGC/---	$\uparrow -2$ AGC/A--	$\uparrow -1$ AGC/AT-	$\nearrow -1$ AGC/ATA
4(A)	$\uparrow -7$ AGCA/-----	$\nearrow -4$ AGCA/---A	$\uparrow -3$ AGCA/AT--	$\nearrow 0$ AGCA/AT-A

# Fazit

## Theorem

*Zur Berechnung eines optimalen paarweisen Alignments zweier Strings der Längen  $n$  und  $m$  genügt Zeit  $O(nm)$  und Platz  $O(nm)$ . Gibt es  $K$  optimale Alignments, so kann man diese in Zeit  $O(K(n + m) + nm)$  aufzählen.*

### Beweis:

- Noch nicht betrachtet:  $K$  optimale Alignments
- Dies lässt sich aber einfach mit mehreren gespeicherten Vorgängerzellen erledigen

Platzbedarf  $(|x| + 1)(|y| + 1)$  problematisch, da  $|x|, |y| \approx 10^6$

## Idee für platzsparendes Vorgehen

**Idee:** Betrachte zusätzlich  $S^r(i, j)$ , die optimale Bewertung eines Alignments zwischen  $(x_{n-i+1}, \dots, x_n)$  und  $(y_{m-j+1}, \dots, y_m)$

Diese Werte lassen sich auch mit dyn. Programmierung berechnen:  
 $(x_n, \dots, x_1)$  und  $(y_m, \dots, y_1)$  statt  $(x_1, \dots, x_n)$  und  $(y_1, \dots, y_m)$

Der Algorithmus verfolgt einen Divide-and-Conquer-Ansatz

Dazu teilen wir ein optimales Alignment an der Stelle  $x_{\lfloor n/2 \rfloor}$  auf, was  $y$  an einer Stelle  $k$  auftrennt.

### Lemma

$$S(n, m) = \max\{S(\lfloor n/2 \rfloor, k) + S^r(\lfloor n/2 \rfloor, m - k) \mid 0 \leq k \leq m\}$$

## Beweis des Lemmas

Sei  $k'$  mit  $0 \leq k' \leq m$  eine Position in  $y$

$S(\lfloor n/2 \rfloor, k')$  ist optimaler Wert für das Paar  
 $(x_1, \dots, x_{\lfloor n/2 \rfloor})$  und  $(y_1, \dots, y_{k'})$

$S^r(\lfloor n/2 \rfloor, m - k')$  ist optimaler Wert für das Paar  
 $(x_{\lfloor n/2 \rfloor + 1}, \dots, x_n)$  und  $(y_{k'+1}, \dots, y_m)$

Es gilt:

$$S(\lfloor n/2 \rfloor, k') + S^r(\lfloor n/2 \rfloor, m - k') \leq S(n, m)$$

Da unabhängig von Wahl von  $k$ :

$$\max\{S(\lfloor n/2 \rfloor, k') + S^r(\lfloor n/2 \rfloor, m - k') \mid 0 \leq k' \leq m\} \leq S(n, m)$$

## Beweis des Lemmas (Fortsetzung)

Rückrichtung ist noch zu zeigen

Seien  $x$  und  $y$  optimal gepaart

Wähle  $k'$  als rechteste Position in  $y$ , die mit einem Zeichen vor  $x_{\lfloor n/2 \rfloor}$  oder  $x_{\lfloor n/2 \rfloor}$  selbst gepaart ist ( $x_{\lfloor n/2 \rfloor} \leftrightarrow -$  möglich)

**Klar:** Summe der Werte von  $(x_1, \dots, x_{\lfloor n/2 \rfloor})$  mit  $(y_1, \dots, y_{k'})$  und  $(x_{\lfloor n/2 \rfloor + 1}, \dots, x_n)$  mit  $(y_{k'+1}, \dots, y_m)$  ist  $S(n, m)$

**Behauptung:** so gewählte Teilpaarungen sind optimal also gleich  $S(\lfloor n/2 \rfloor, k')$  bzw.  $S^r(\lfloor n/2 \rfloor, m - k')$

Andernfalls wäre die Summe der optimalen Teilpaarungen größer  $S(n, m)$

Widerspruch zu

$$\max\{S(\lfloor n/2 \rfloor, k') + S^r(\lfloor n/2 \rfloor, m - k') \mid 0 \leq k' \leq m\} \leq S(n, m)$$

# Algorithmus von Hirschberg

## Algorithmus von Hirschberg

- 1 In Zeit  $O(nm)$  und Platz  $O(m)$  werden die  $S(\lfloor n/2 \rfloor, \cdot)$ -Werte berechnet und abgespeichert.
- 2 In Zeit  $O(nm)$  und Platz  $O(m)$  werden die  $S^r(\lfloor n/2 \rfloor, \cdot)$ -Werte berechnet und abgespeichert.
- 3 In Zeit  $O(m)$  wird das  $k^*$ , das  $S(n, m) = \max\{S(\lfloor n/2 \rfloor, k) + S^r(\lfloor n/2 \rfloor, m - k) \mid 0 \leq k \leq m\}$  maximiert, berechnet. Es wird  $(\lfloor n/2 \rfloor, k^*)$  abgespeichert, die Ergebnisse der ersten beiden Schritte werden gelöscht.
- 4 Löse rekursiv die Probleme für die Folgen  $(x_1, \dots, x_{\lfloor n/2 \rfloor})$  und  $(y_1, \dots, y_{k^*})$  sowie für die Folgen  $(x_{\lfloor n/2 \rfloor + 1}, \dots, x_n)$  und  $(y_{k^* + 1}, \dots, y_m)$  und konkateniere die Lösungen. Die Rekursion endet in den trivialen Fällen, nämlich, wenn eine Folge leer wird oder eine Folge nur noch Buchstaben enthält.

# Laufzeit, Speicherplatz, Korrektheit

## Theorem

*Der Algorithmus von Hirschberg löst das Problem des globalen Alignments konstruktiv in Zeit  $O(nm)$  auf Platz  $O(n + m)$*

### Beweis:

- Korrektheit ist nach Vorüberlegungen klar ✓
- Platzbedarf:
  - In jedem Rekursionsaufruf nur  $O(1)$
  - $O(n + m)$  Rekursionsaufrufe ✓
- Laufzeit?

# Laufzeit

Wähle Konstante  $c$  so, dass

- Laufzeit der Schritte 1-3  $cnm$
- Laufzeit triviale Fälle  $cn$  bzw.  $cm$

O.B.d.A.  $n$  Zweierpotenz (sonst Auffüllen mit Buchstaben, der mit Leerzeichen gepaart werden muss)

Sei  $T(n, m)$  Zeitschranke:

$$T(n, m) \leq cnm + \max\{T(n/2, k) + T(n/2, m-k) \mid 0 \leq k \leq m\}$$

Rate  $T(n, m) \leq 2cnm$  (korrekt für triviale Fälle)

$$\begin{aligned} T(n, m) &\leq cnm + \max\left\{2c\frac{n}{2}k + 2c\frac{n}{2}(m-k) \mid 0 \leq k \leq m\right\} \\ &= cnm + \max\{cn(k + (m-k)) \mid 0 \leq k \leq m\} \\ &= 2cnm \end{aligned}$$



# Semiglobales Alignment

- Bisher *globales* Alignment:  $x$  und  $y$  von ähnlicher Länge
- Bei stark unterschiedlicher Länge entstehen Alignments mit vielen Leerzeichen im kürzeren String
- Angenommen  $|x| \ll |y|$

## Ideen:

- Bewerte Lücken vor dem Anfang und nach dem Ende von  $x$  nicht
- Bewerte Lücken “in”  $x$  negativ
- Benutze Algorithmus von Hirschberg

## Ausführung:

- Lücken vor dem Anfang:  $S(0, \cdot) = 0, S(\cdot, 0) = 0$
- Lücken nach dem Ende: Wähle statt  $S(n, m)$  das maximale aus  $S(n, \cdot)$  und  $S(\cdot, m)$

# Lokales Alignment

## Definition

Seien  $x = x_1 \cdots x_n$ ,  $y = y_1 \cdots y_m$  Strings über einem Alphabet  $\Sigma$ .  
Finde  $i, i', j, j'$  mit  $0 \leq i \leq i' \leq n$  und  $0 \leq j \leq j' \leq m$ , so dass die  
Bewertungen des globalen paarweisen Alignments der Substrings  
 $x_i x_{i+1} \cdots x_{i'}$  und  $y_j y_{j+1} \cdots y_{j'}$  maximal ist.

Abwandlung des Needleman-Wunsch-Algorithmus:

$$\begin{cases} s(a, b) \leq 0 & \text{falls } a = - \text{ oder } b = - \text{ oder } a, b \text{ nicht passen.} \\ s(a, b) \geq 0 & \text{falls } a, b \text{ passen} \end{cases}$$

Wie beim semiglobalen Alignment:  $S(0, \cdot) = 0, S(\cdot, 0) = 0$

Wir schließen negative Werte  $S(i, j)$  aus, was dem Start einer neuen Teilfolge entspricht.

# Lokales Alignment (Fortsetzung)

## Bellmannsche Optimalitätsgleichung

$$S(i, j) = \max\{0, S(i-1, j-1) + s(x_i, y_j), S(i-1, j) - d, S(i, j-1) - d\}$$

Wert des lokalen Alignments ist dann:

$$S_{\max} = \max\{S(i, j) \mid 0 \leq i \leq n, 0 \leq j \leq m\}$$

# Affine Lückenkosten

- Extrastrafe für die Eröffnung einer Lücke
- Biologisch motiviert
- Lückenkosten sind  $-c - d\ell$ , wobei  $\ell$  die Länge der Lücke ist und  $c$  der zusätzliche Strafterm
- Damit Abhängigkeit der Bewertung von mehreren vorhergehenden Stellen

## Idee: Aufteilung in vier Fälle

- $A(i, j)$ : beste Bewertung, wenn am Ende  $x_i$  mit  $y_j$  gepaart ist
- $B(i, j)$ : beste Bewertung, wenn am Ende  $x_i$  mit  $-$  gepaart ist
- $C(i, j)$ : beste Bewertung, wenn am Ende  $-$  mit  $y_j$  gepaart ist
- $D(i, j)$ : beste Bewertung ohne Einschränkung

# Gleichungen für affine Lückenkosten

## Vier Parameter:

$A(i, j)$  ( $x_i \leftrightarrow y_j$ ),  $B(i, j)$  ( $x_i \leftrightarrow -$ ),  $C(i, j)$  ( $- \leftrightarrow y_j$ ),  $D(i, j)$

$$A(i, 0), A(0, j) = -\infty$$

$$B(0, j), C(i, 0) = -\infty$$

$$B(i, 0) = -c - di$$

$$C(0, j) = -c - dj$$

$$D(i, 0) = \max\{A(i, 0), B(i, 0), C(i, 0)\} = B(i, 0)$$

$$D(0, j) = \max\{A(0, j), B(0, j), C(0, j)\} = C(0, j)$$

$$A(i, j) = D(i - 1, j - 1) + s(x_i, y_j)$$

$$B(i, j) = \max\{B(i - 1, j) - d, D(i - 1, j) - c - d\}$$

$$C(i, j) = \max\{C(i, j - 1) - d, D(i, j - 1) - c - d\}$$

$$D(i, j) = \max\{A(i, j), B(i, j), C(i, j)\}$$