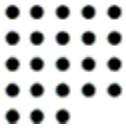


Fachhochschule Köln
University of Applied Sciences Cologne

Algorithmische Anwendungen
WS 2005/2006

Sequenzalignment

Gruppe F_lila_Ala0506

Allal Kharaz
Yassine ELassad

Inhaltsverzeichnis

1 Problemstellungen

In der Informatik gibt es viele Problemstellungen, die fehlertolerante Mustervergleiche erfordern. Hierunter fallen die Spracherkennung und der Vergleich von Fingerabdrücken.

1.1 Rechtschreibkorrektur

Eines der bekanntesten Beispiele für solche Problemstellungen ist die Rechtschreibkorrektur wie man sie heute in modernen Textverarbeitungsprogrammen wie Microsoft Word findet. Hier wird z.B. für DNA als Wort DANN vorgeschlagen.

1.2 DNA- und Aminosäure-Sequenzen

Für Biologen sind besonders Sequenzuntersuchungen ein interessantes Problemfeld. So lassen sich Protein- oder DNA-Sequenzen, die im Labor entdeckt wurden, hinsichtlich ihrer Sequenz¨ahnlichkeit katalogisieren und geben so Aufschluß über ihre Funktionalität.

2 Was ist ein Alignment?

Abstrahiert man eine biologische Sequenz als einen eindimensionalen String, so kann der fehlertolerante Mustervergleich in der Molekularbiologie als zeichenweiser Vergleich von zwei Strings verstanden werden. Ziel dieses Vergleiches ist eine Aussage über den Grad der Abweichung der beiden Strings.

2.1 Globales Alignment

In einem globalen Alignment werden die Sequenzen so durch Einfügen von "-" verlängert, dass möglichst viele Sequenzteile die übereinstimmen in beiden Verlängerungen an der selben Position stehen. Es werden dabei immer die beiden kompletten Sequenzen betrachtet. Will man Beispielsweise die Sequenzen **"ANNA"** und **"HANA"** vergleichen, ist unter vielen anderen auch folgendes Alignment möglich:

```
-ANNA
HAN-A
```

2.1.1 Definition

• Ein globales Alignment von A und H sind zwei Strings A' und H' mit

- E ... endliches Alphabet, E' = E ∪ "-"
- $A = a_1 + a_2 + \ldots + a_n$ mit $a_i \in E$
- $H = h_1 + h_2 + \ldots + h_m$ mit $h_i \in E$
- $A' = a'_1 + a'_2 + \ldots + a'_k$ mit $a'_i \in E'$
- $H' = h'_1 + h'_2 + \ldots + h'_k$ mit $h'_i \in E'$
- $n, m \leq k \leq n + m$

– Entfernt man alle "-" aus H', A' erhält man H, A

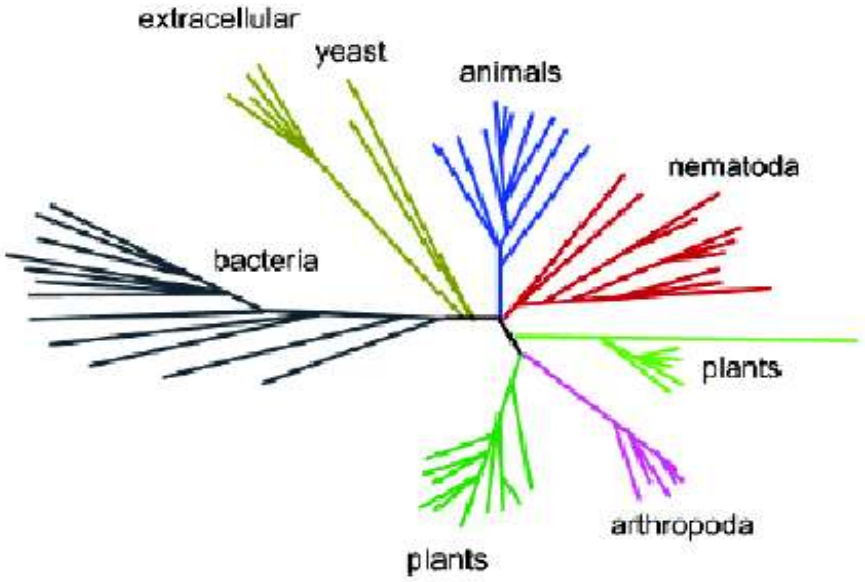
Als Folge dieser Definition ergibt sich, dass es viele Möglichkeiten gibt zwei beliebige Sequenzen zu alignieren.
Um eine Aussage über die Güte der gerade untersuchten Anordnung zu geben, müssen die Sequenz-Alignments bewertet werden. Dies geschieht wie weiter unten beschrieben durch Einführung einer "Scoring-Funktion".

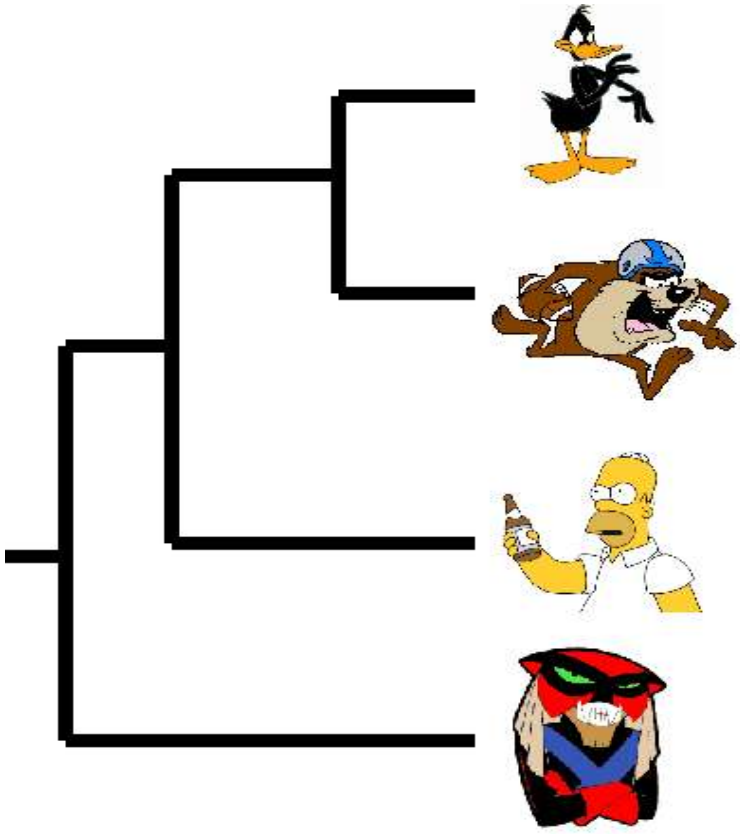
2.2 Lokales Alignment

Proteine sind oftmals aus einem Repertoire ausähnlichnen Untereinheiten aufgebaut, obwohl sie global wenig Ähnlichkeit zeigen. Die Suche nach gemeinsamen Untereinheiten kann mittels lokalem Alignment erfolgen.
Ein optimales lokales Alignment ist also ein optimales globales Alignment über die Substrings von A und H. Das heißt es gibt kein Paar von Substrings von A und H das einen höheren Alignment-Score hat.

3 Anwendungsgebiete:

Sequenzealignment wird besonders häufig in der Bioinformatik verwendet, um die funktionelle oder evolutionäre Verwandtschaft (Homologie) von DNA- oder Proteinsequenzen zu untersuchen.





4 Transformationsoperationen

Um die beiden gegebenen Sequenzen an einander auszurichten, bedarf es einiger Operationen.
Hierbei werden die drei Operationen **Deletion**, d.h das Entfernen eines Zeichens aus A, **Insertion** - das Einfügen eines Zeichens in H und die **Substitution** - das Ersetzten eines Zeichens verwendet.

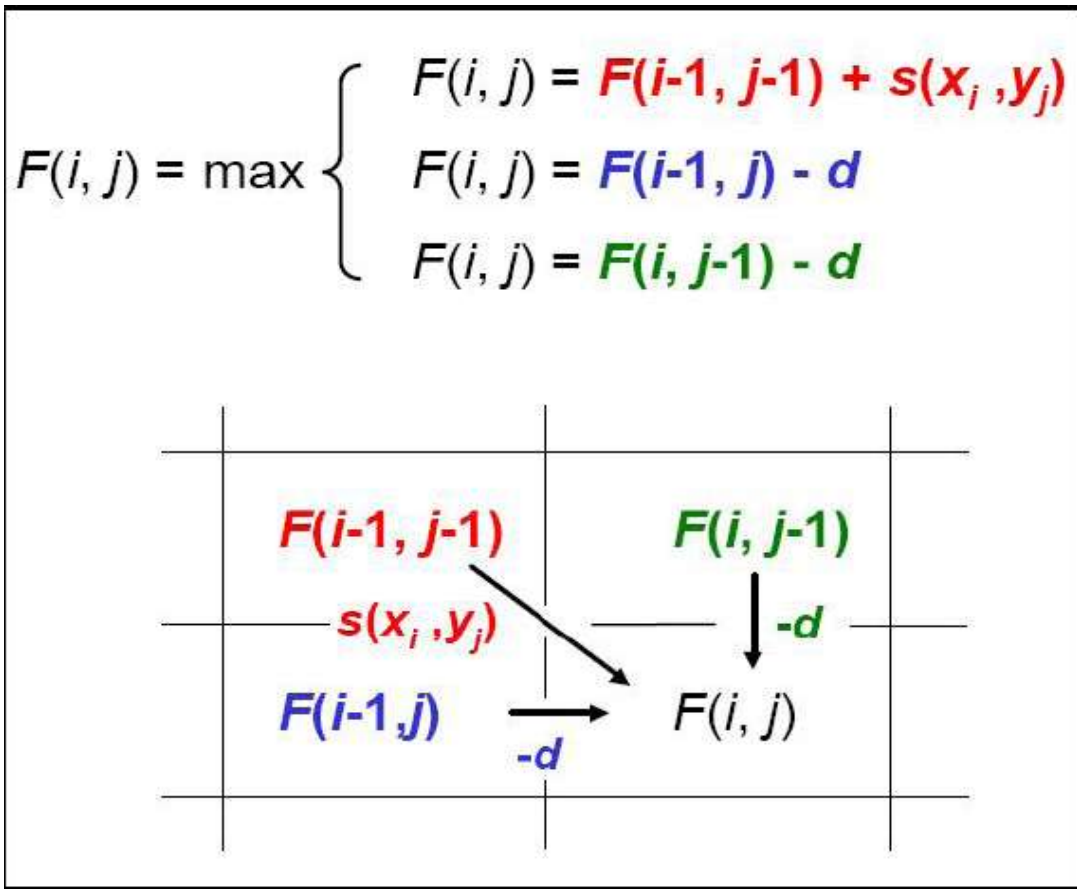
Deletion	***Insertion***	***Substitution***
Löschung von ai in A --> Einfügen von "-" in H'	Einfügen eines Buchstaben an hi. --> Einfügen von "-" in A'	Ersetzen von hj durch ai

Das folgende Beispiel veranschaulicht die drei Operationen noch einmal grafisch:

4.1 Alignment-Matrix

Das optimale globale Aligment kann man relativ leicht aus der folgendermaßen rekursiv definierten Matrix ableiten:

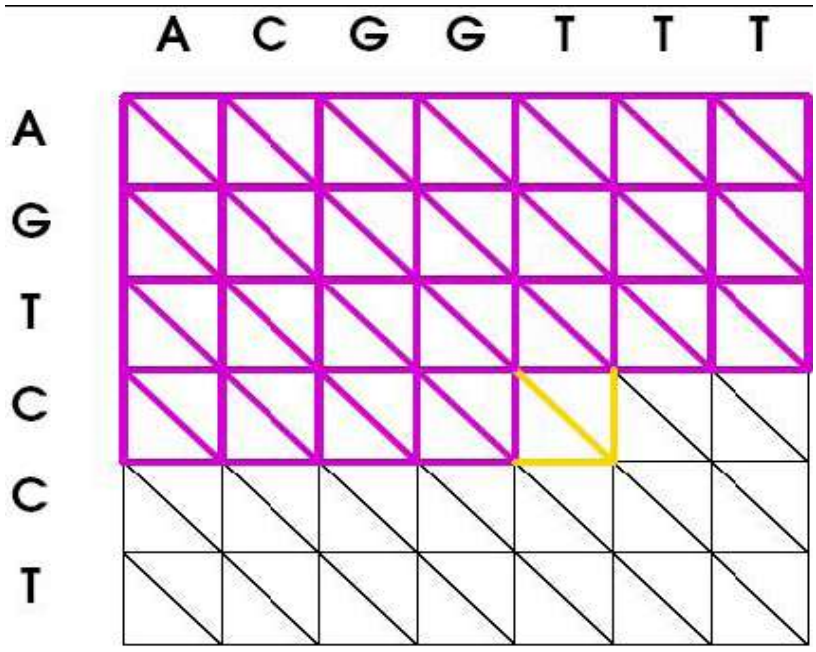
PS. d = S(Xi , -) oder S(- , Yi)



Implementiert man dies jedoch direkt auf diese Weise, so ist die Berechnung sehr aufwändig.
Abhilfe schafft hier die dynamische Programmierung deren Idee es ist an einem Ende der Sequenz zu starten und dann solange einzelne Symbole zur aktuell besten Lösung hinzuzufügen bis alle Symbole verbraucht sind.

4.2 Needleman-Wunsch-Algorithmus - Globales Alignment

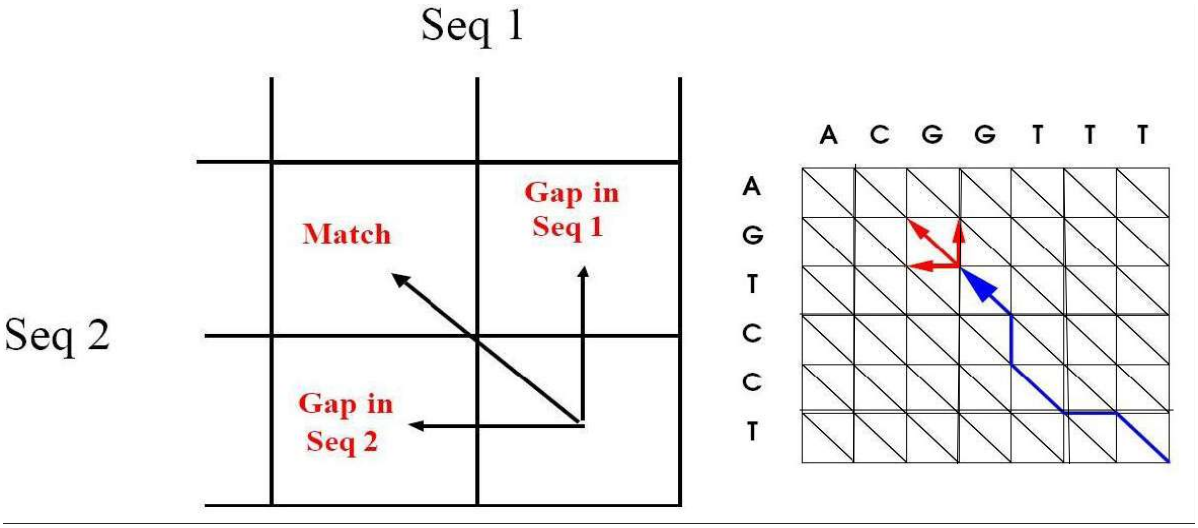
Ein spezielle Implementierung dieser Idee liefert der **Needleman-Wunsch-Algorithmus**.



• Man berechne Fij Zeile für Zeile von links nach rechts
• Ergebnisse zwischenspeichern
• Es stehen die benötigten Werte immer rechtzeitig zur Verfügung

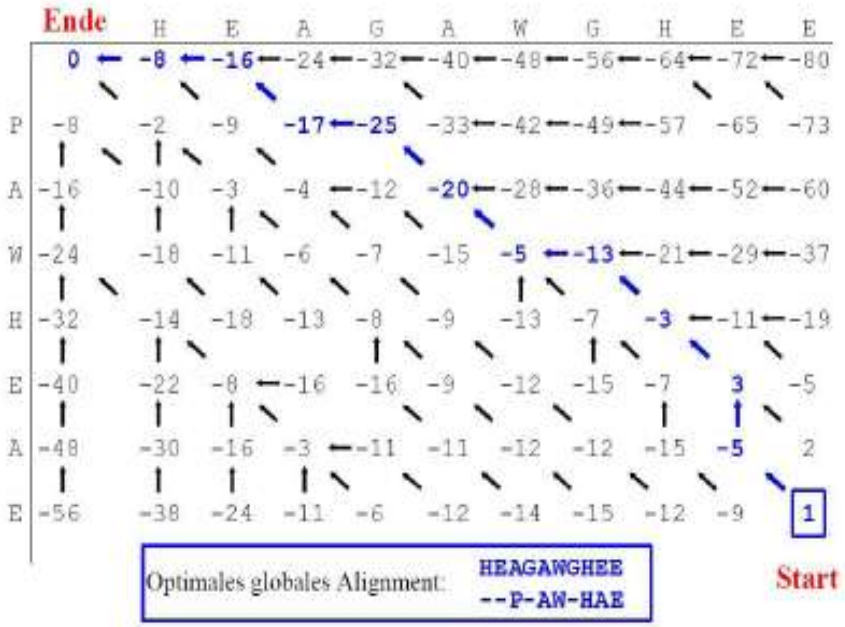
4.3 Traceback

Am Ende der Zellenberechnung kann man die optimalen Alignments durch ein sogenanntes Traceback aus der Alignment-Matrix ableiten. Dazu läuft man einfach vom Feld F(i, j) durch Matrix und geht in das Feld zurück, dass das Maximum im Needleman-Wunsch Algorithmus geliefert hat. Man findet so alle optimalen Alignments in linearer Laufzeit.



4.4 Bespiel globales Alignment

- Globales Alignment **HEAGAWGHEE** mit **PAWHEAE**
- Score-Matrix BLOSUM50
- Linearer Gap (d = 8)



4.5 Smith-Waterman-Algorithmus - Lokales Alignment

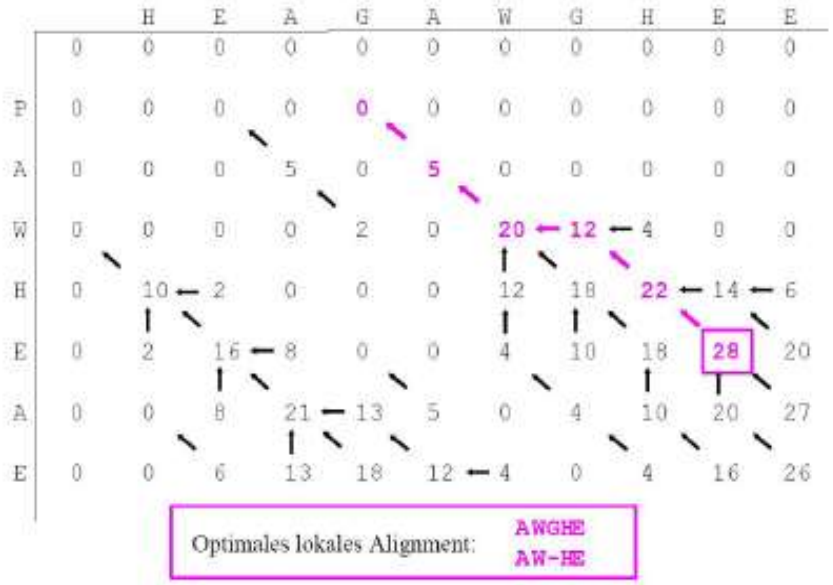
Der Smith-Waterman-Algorithmus ist eine ähnliche Variante, jedoch diesmal zugeschnitten auf lokale Alignments. Er findet somit den längsten Bereich zweier Sequenzen mit der größten ähnlichkeit. Die Berechnung erfolgt völlig analog zu Needleman-Wunsch. Jedoch mit der Besonderheit, dass wenn alle drei Alignment möglichkeiten einen negativen Score liefern, der Score-Wert wieder mit Null initialisert wird und die Kette abgebrochen wird. Als Folge daraus ergibt sich, dass Pfade direkt in der Matrix starten und enden können.

Es ergibt sich also folgende rekursive Definition:

$$F(i,j) = \max \begin{cases} 0 \\ F(i\text{-}1, j\text{-}1) + s(x_i, y_j) \\ F(i,j) = F(i\text{-}1, j) - d \\ F(i,j) = F(i, j\text{-}1) - d \end{cases}$$

4.6 Bespiel lokales Alignment

• Lokales Alignment **HEAGAWGHEE** mit **PAWHEAE**
• Score-Matrix BLOSUM50
• Linearer Gap (d = 8)



5 Komplexität

Der **Needleman-Wunsch-Algorithmus** und **Smith-Waterman-Algorithmus** haben eine Laufzeitkomplexität von $O(n^2)$. Diese Laufzeit entsteht, da der Algorithmus jedes der $n * m$ Felder mit einer Bewertung belegen muss. Die Berechnung der Bewertung funktioniert in der Regel in $O(1)$.

Die Speicherplatzkomplexität liegt ebenfalls bei $O(n^2)$, da alle Werte der Feldes abgespeichert werden.

Der Algorithmus ist daher für lange Alignments eher ungeeignet. Wenn man davon ausgeht, dass man Integerwerte speichert und diese in Java 4 Byte Speicher belegen, kommt man bei einem Alignment von 10.000 * 10.000 Buchstaben schon auf 100.000.000 * 4 Byte $\approx$ 381 MegaByte.