

Bioinformatik

Ähnlichkeit von Zeichenketten



Ulf Leser
Wissensmanagement in der
Bioinformatik



Motivation

- BLAST / FASTA und Verwandte sind *die* Bioinformatik Anwendung
 - Teilweise synonym für „Bioinformatik“
- Grundlegende Gesetzmäßigkeit der Bioinformatik

Hohe Sequenzähnlichkeit bei DNA, RNA und Proteinen heißt in der Regel ähnliche Funktion bzw. Struktur

Begründung

- Biochemische Aktivität der Proteine wird bestimmt durch
 - 3D Faltung der Proteine
 - Vorkommen bestimmter Aminosäuren an bestimmten Stellen
 - Interaktion und Modifikation von Proteinen
- **Wesentliches Element ist die 3D Struktur von Proteinen**
 - Die vor allem von der Proteinsequenz und damit von der DNA Sequenz des Gens abhängt
- Durch Evolution entstehen Variationen
 - Funktionale Änderungen schaffen Varianten (oder sind letal)
 - Evolution geht meist in **kleinen Schritten**: Kleine Änderungen, leicht veränderte Struktur, leicht verändertes funktionales Verhalten
 - Stammbaumentstehung (Phylogenie)
- Bestimmung von Funktion ist extrem aufwändig (wenn überhaupt möglich), Bestimmung von Sequenzen dagegen sehr billig
- **Also: Annäherung der Funktion über Sequenzähnlichkeiten**

Beispiel

- „AGGTAG“ in „AGTAGGTTAGGATAGTTCAGA“
 - 1: AGTAGGTTAGGATAGCTCAGA
 - 2: AGTAGGTTAGGATAGCTCAGA
 - 3: AGTAGGTTAGGATAGCTCAGA
 - 4: AGTAGGTTAGGATAGTTCAGA
- Welche Matches sind besser?
 - 1: „G“ fehlt
 - 2: Perfekt
 - 3: „A“ zuviel
 - 4: „T“ durch „G“ ersetzen oder „T“ löschen und „G“ einfügen, ein „C“ zuviel

Comparative Genomics

- Bestimmung von Protein/Genfunktion in anderen Spezies wesentlich leichter als beim Menschen
 - Bakterien, Knock-out Mäuse, etc.
- Viele Gene sind hochgradig konserviert
 - Maus – Mensch: 97% Sequenzidentität
 - „Housekeeping Genes“ in allen Organismen ähnlich vorhanden
 - Die 4% „aktivsten“ (am besten verbundenen) Proteine sind in allen (bisher sequenzierten) bekannten Organismen vorhanden
- Vorwärts
 - Finden und sequenzieren eines neues Genes beim Menschen
 - Suchen nach ähnlichen Sequenzen in anderen Organismen
 - [BLAST gegen Genbank / EMBL](#)
- Rückwärts
 - Bestimmung der Funktion eines Genes einer anderen Spezies
 - Suche nach ähnlichen Sequenzen beim Menschen
 - [BLAST gegen Genbank / EMBL](#)



Approximatives Matchen außerhalb der Bioinformatik

- Anwendungen außerhalb der Bioinformatik
 - Unschärfe Volltextsuche
 - Suche mit „Xylofon“ und finde auch „Xhylophon“
 - Personenabgleich
 - Ist „Herr Müller, 27, Stargarder Str 54“ identisch zu „Hr. Mueller, 27, Stagarder Str. 54“ ?
 - Phonetische Suche
 - Finde alle Meyer, Meier, Maier, Mair, ...



Dotplot

- Definition

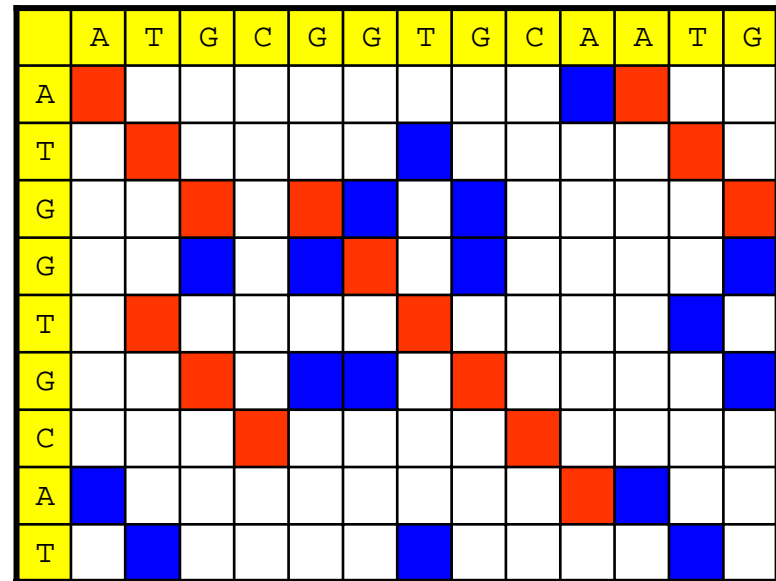
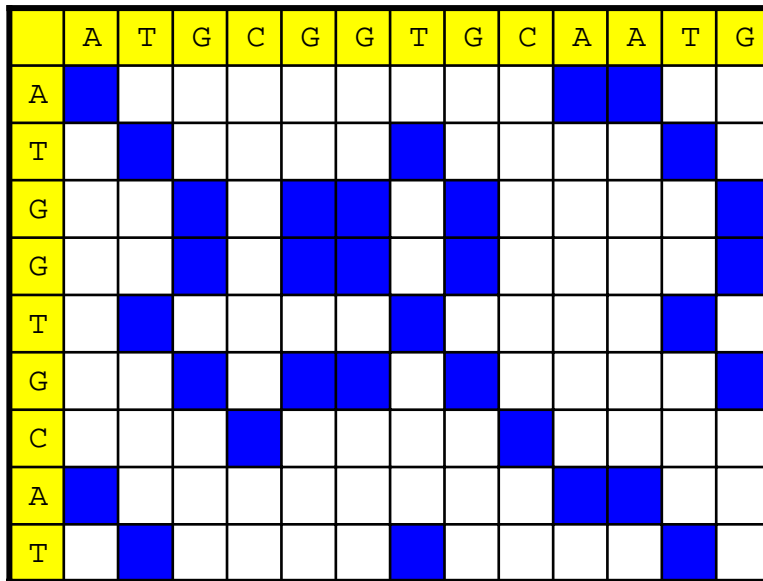
Ein *Dotplot* zweier Strings A , B ist eine Matrix M mit

- Die Spalten entsprechen den Zeichen von A
- Die Zeilen entsprechen den Zeichen von B
- $M[a,b]=1$ gdw. $A[a] = B[b]$; sonst 0

	A	T	G	C	G	G	T	G	C	A	A	T	G
A	1	0	0	0	0	0	0	0	0	1	1	0	0
T	0	1	0	0	0	0	1	0	0	0	0	1	0
G	0	0	1	0	1	1	0	1	0	0	0	0	1
G	0	0	1	0	1	1	0	1	0	0	0	0	1
T	0	1	0	0	0	0	1	0	0	0	0	1	0
G	0	0	1	0	1	1	0	1	0	0	0	0	1
C	0	0	0	1	0	0	0	0	1	0	0	0	0
A	1	0	0	0	0	0	0	0	0	1	1	0	0
T	0	1	0	0	0	0	1	0	0	0	0	1	0

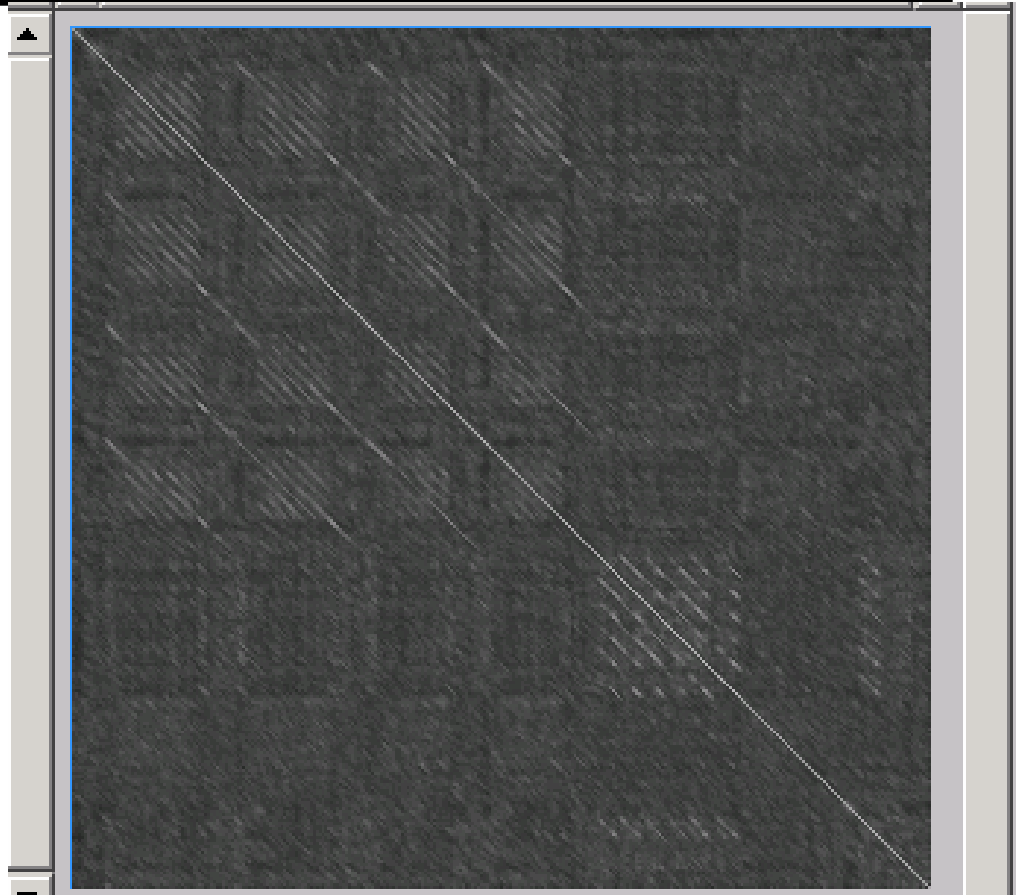
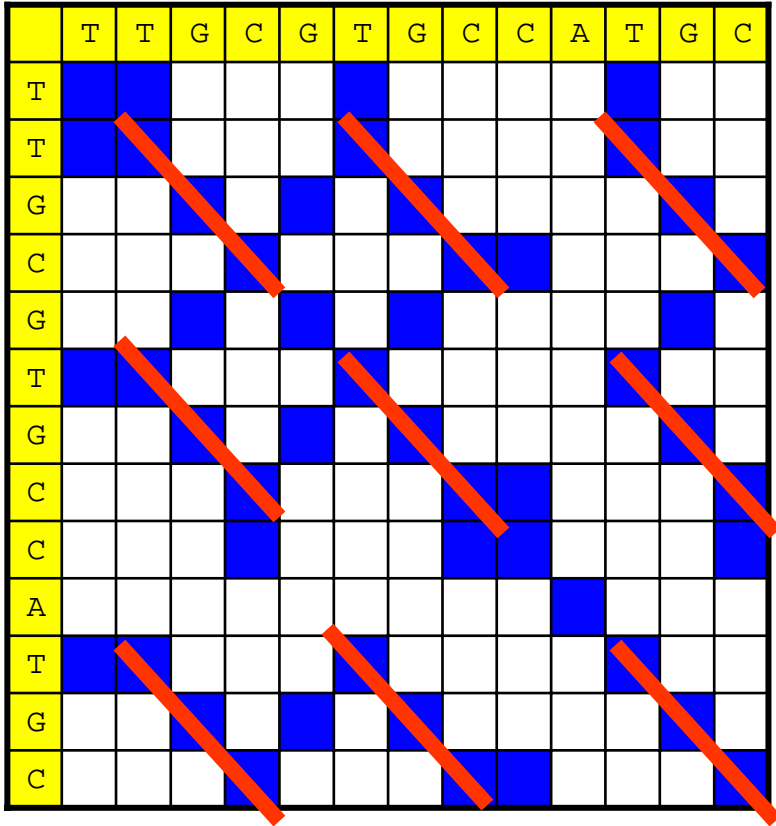
Dotplot und gleiche Teilstrings

- Wie erkennt man gleiche Teilstrings im Dotplot?



- Diagonalen von links-oben nach rechts-unten
 - Größter gemeinsamer Teilstring – längste Diagonale
 - Visuell bei kurzen Strings möglich

Repetitive Sequenzen



- Dotplot mit $A=B$

– Zitat (Genbank, P24014):

[SIMILARITY] CONTAINS 7 EGF-LIKE DOMAINS.

[SIMILARITY] Contains 24 leucine-rich (LRR) repeats.

Abstandsmaße

- Approximatives Stringmatching sucht Ähnlichkeiten
 - Welcher Substring von T ist am ähnlichsten zu P?
 - Welcher String T_1, \dots, T_n ist am ähnlichsten zu T?
- Voraussetzung dafür
 - Was heißt ähnlich?
 - Was heißt „am ähnlichsten“?
- Quantifizierung des Abstandes zweier Strings
 - Definition von Ähnlichkeit ist oft eine sehr schwierige Aufgabe
 - Ähnlichkeit ist abhängig vom Gegenstand und Aufgabe
 - Wann sind sich Gesichter ähnlich - Haarfarbe zählt weniger als Augenfarbe?
 - Wann sind sich Texte ähnlich – gleiche Wörter oder gleicher Inhalt?

Mögliche Maße

- Hammingabstand
 - Voraussetzung: $|A|=|B|$
 - Vergleiche A und B Zeichen für Zeichen
 - Hammingabstand = Anzahl der Mismatches
 - Beispiel: $ha(\text{CGTGCTCGC}, \text{ACGTGCTCG}) = 9$
 - Das kann nicht in unserem Sinne sein
- Biologischen Hintergrund nicht vergessen
 - Situation: Wir haben humane Gensequenz A und suchen ähnliche Sequenzen (B) in anderen Organismen
 - Annahme: A und B haben gemeinsamen Vorfahren und sind durch **evolutionäre Prozesse** entstanden
 - Welche Annahme steckt hier drin?
 - Einfaches Modell: **Basenaustausch, Baseneinfügung, Basenlöschungen**

Begriffe

- Sequenzen heißen
 - **Homolog**, wenn sie einen gemeinsamen Ursprung haben und von diesem durch Evolution divergiert sind
 - **Ortholog**, wenn sie in verschiedenen Spezies vorkommen, aber vom gleichen „Vorfahren“ abstammen
 - **Paralog**, wenn sie durch Duplikation innerhalb einer Spezies entstanden sind
- Ob zwei ähnliche Sequenzen homolog, paralog oder ortholog sind (oder weder noch), kann man eigentlich nicht bestimmen
 - Nur Indizien sammeln
 - Ähnlichkeit der Sequenz ist ein sehr starkes Indiz
 - Andere: Lage im Genom, Regulationsmechanismen, Beteiligung in den gleichen Stoffwechselwegen an gleicher Stelle, ...

Editskripte

- Definition

Ein *Editskript* e für zwei Strings A, B aus $\Sigma^* = \Sigma \cup \{ "_\}$ ist eine Sequenz von Editieroperationen

- I (*Einfügen* eines Zeichen $c \in \Sigma$ in A)
 - Dargestellt als Lücke in A ; das neue Zeichen erscheint in B
- D (*Löschen* eines Zeichen c in A)
 - Dargestellt als Lücke in B ; das alte Zeichen erscheint in A
- R (*Ersetzen* eines Zeichen in A mit einem anderen Zeichen in B)
- M (*Match*, d.h., gleiche Zeichen in A und B an dieser Stelle)

so, dass $e(A) = B$

- Beispiel: $A = \text{„ATGTA“}$, $B = \text{„AGTGTC“}$

– MIMMMR	IRMMMDI
A_TGTA	_ATGTA_
AGTGTC	AGTGT_C

Editabstand

- Offensichtlich gibt es für A, B ziemlich viele Editskripte
 - Wie viele?
- Definition
 - Die *Länge eines Editskript* ist die Anzahl von Operationen o im Skript mit $o \in \{I, R, D\}$
 - Der *Editabstand* zweier Strings A, B ist die Länge des kürzesten Editskript für A, B
- Bemerkung
 - Matchen zählt nicht – interessant sind nur die Änderungen
 - Anderer Name: **Levenshtein-Abstand**
 - Es gibt oft verschiedene kürzeste Editskripte
 - | | |
|---------|---------|
| IMMMMMD | DMMMMMI |
| _AGAGAG | AGAGAG_ |
| GAGAGA_ | _GAGAGA |

Alignment

- Andere Darstellung: **Alignments**
- Definition
 - Ein (*globales*) **Alignment** zweier Strings A, B ist eine Untereinanderanordnung von A und B , jeweils mit beliebigen zusätzlichen Leerzeichen (`_`), ohne dass zwei Leerzeichen untereinander stehen
 - Achtung: Untereinanderstehende Zeichen müssen nicht matchen
 - Der **Alignmentsscore** eines Alignment ist die Anzahl von Leerzeichen und Mismatches
 - Der **Alignmentabstand** zweier Strings A, B ist der minimale Alignmentsscore aller Alignments der beiden Strings

- Beispiele

– A_TGT_A	A_T_GTA	_AGAGAG	AGAGAG_
AGTGTC_	_AGTGTC	GAGAGA_	_GAGAGA

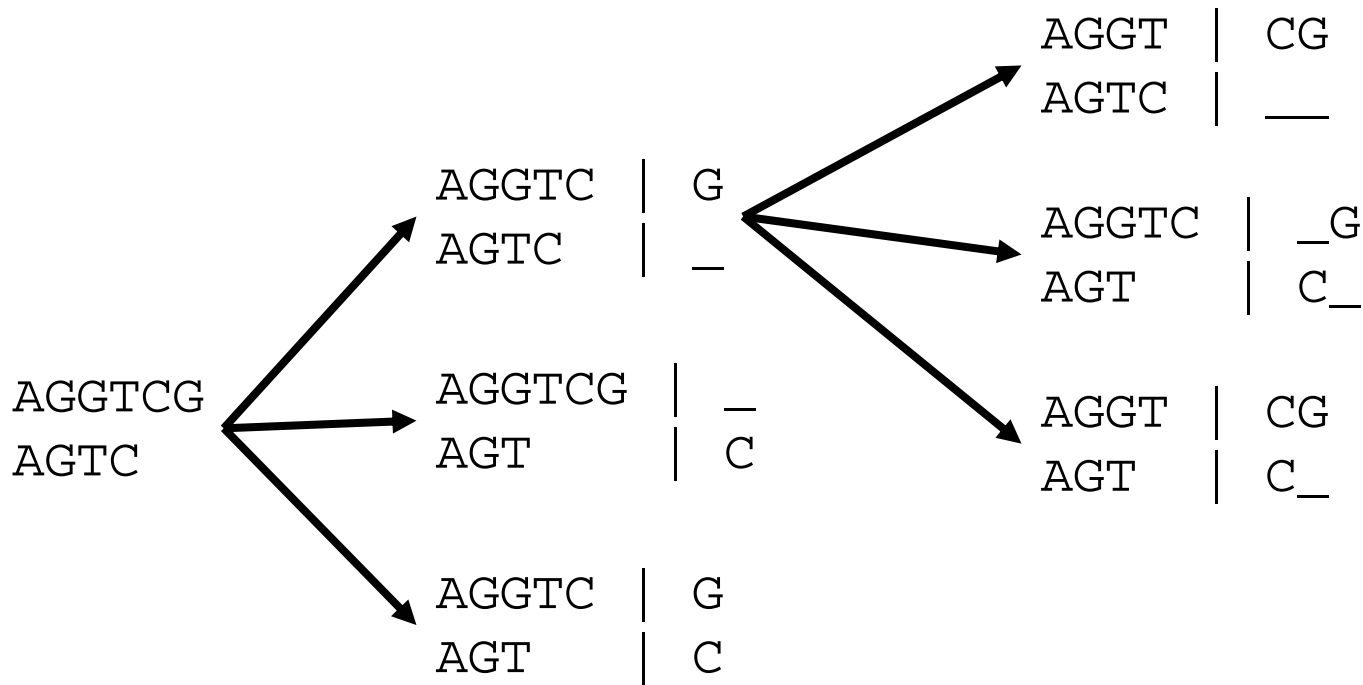
Score: **3**

5

2

2

Rekursive Definition von Alignments



Editabstände

- Definition

Gegeben zwei Strings A, B mit $|A|=n, |B|=m$

- *Funktion $\text{dist}(A,B)$ berechne den Editabstand von A, B*
- *Funktion $d(i,j)$, $0 \leq i \leq n$ und $0 \leq j \leq m$, berechne den Editabstand zwischen $A[1..i]$ und $B[1..j]$*

- Bemerkungen

- Jedes R kann durch $\{I,D\}$ ersetzt werden; also werden R bevorzugt
- Offensichtlich gilt: $d(n,m) = \text{dist}(A,B)$
- $d(i,j)$ dient zur rekursiven Berechnung von $\text{dist}(A,B)$

Rekursive Berechnung 1

- Wir betrachten die Berechnung von $d(i,j)$ für A,B
 - Wir haben die optimalen Editskript für $A[1..i_0]$ mit $B[1..j_0]$, $i_0 \leq i \wedge j_0 \leq j \wedge \neg(i_0 = i \wedge j_0 = j)$, berechnet
 - Wie kann das Editskript weitergeführt werden?
- Fallunterscheidung
 - 1. Insertion in A (oder Deletion in B)
 - Situation:
 ...I
 XXX_
 XXXT
 - Also benutzen wir ein Zeichen mehr von B
 - $d(i,j-1)$ ist der Editabstand von $A[1..i]$ zu $B[1..j-1]$
 - Symbolisiert durch die XXX
 - Damit: $d(i,j) = d(i, j-1) + 1$

Rekursive Berechnung 2

- Wir betrachten die Berechnung von $d(i,j)$ für A,B
 - Wir haben die optimalen Editskript für $A[1..i_0]$ mit $B[1..j_0]$, $i_0 \leq i \wedge j_0 \leq j \wedge \neg(i_0 = i \wedge j_0 = j)$, berechnet
 - Wie kann das Editskript weitergeführt werden?
- Fallunterscheidung
 - 2. Deletion in A (oder Insertion in B)
 - Situation:
 D
 XXXXT
 XXX_
 - Umgekehrte Situation
 - Wir benutzen ein Zeichen mehr von A
 - $d(i-1, j)$ ist der Editabstand von $A[1..i-1]$ zu $B[1..j]$
 - Damit: $d(i, j) = d(i-1, j) + 1$

Rekursive Berechnung 3

- Wir betrachten die Berechnung von $d(i,j)$ für A,B
 - Wir haben die optimalen Editskript für $A[1..i_0]$ mit $B[1..j_0]$, $i_0 \leq i \wedge j_0 \leq j \wedge \neg(i_0 = i \wedge j_0 = j)$, berechnet
 - Wie kann das Editskript weitergeführt werden?
- Fallunterscheidung
 - 3. Match
 - Situation:
 . . . M
 XXX Y
 XXX Y
 - Wir benutzen ein Zeichen mehr von A und eines mehr von B
 - Match kostet nichts
 - Damit: $d(i,j) = d(i-1, j-1)$

Rekursive Berechnung 4

- Wir betrachten die Berechnung von $d(i,j)$ für A,B
 - Wir haben die optimalen Editskript für $A[1..i_0]$ mit $B[1..j_0]$, $i_0 \leq i \wedge j_0 \leq j \wedge \neg(i_0 = i \wedge j_0 = j)$, berechnet
 - Wie kann das Editskript weitergeführt werden?
- Fallunterscheidung
 - 4. Mismatch
 - Situation:
 ...R
 XXXY
 XXXZ
 - Wir benutzen ein Zeichen mehr von A und eines mehr von B
 - Mismatch kostet 1
 - Damit: $d(i,j) = d(i-1, j-1) + 1$

Rekursionsgleichung

- Wir leiten das nächste Symbol im Editskript aus schon bekannten Editabständen ab
- Wir suchen das kürzeste Skript, also

$$d(i, j) = \min \left\{ \begin{array}{l} d(i, j-1) + 1 \\ d(i-1, j) + 1 \\ d(i-1, j-1) + t(i, j) \end{array} \right\}$$

$$t(i, j) = \begin{cases} 1: & \text{wenn } A[i] \neq B[j] \\ 0: & \text{sonst} \end{cases}$$

Randbedingungen

- Randbedingungen nicht vergessen
 - $d(i,0) = i$
 - Um $A[1..i]$ zu „“ zu transformieren braucht man i Deletions
 - $d(0,j) = j$
 - Um $A[1..0]$ zu $B[1..j]$ zu transformieren braucht man j Insertions

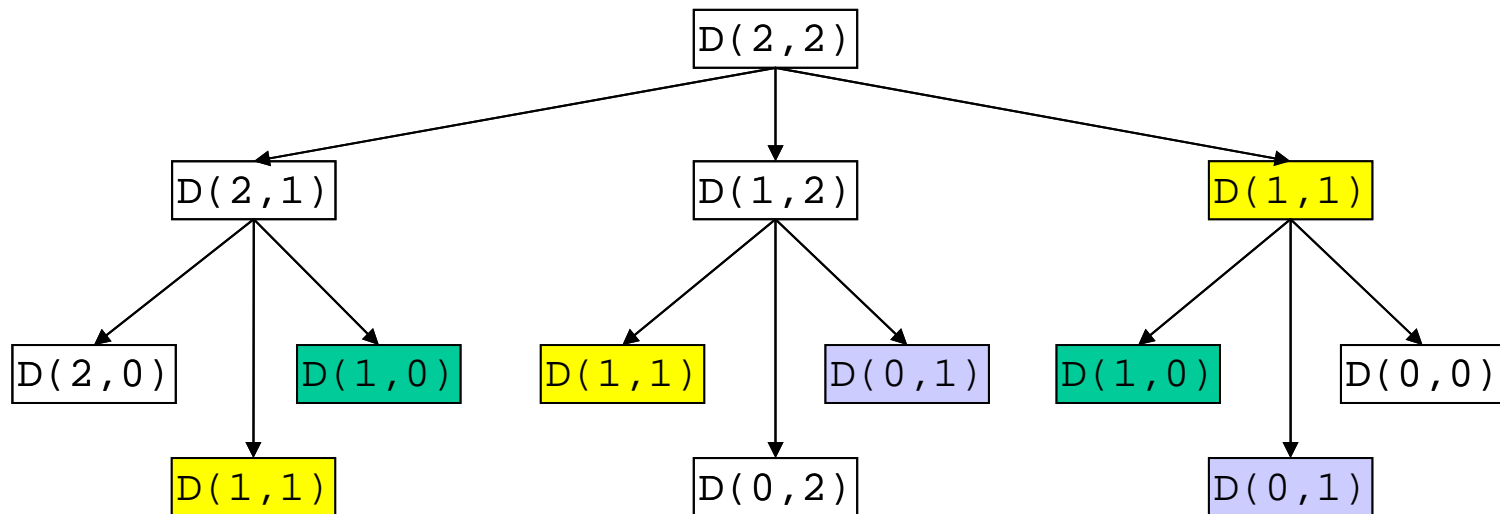
Rekursiver Algorithmus

```
function d(i,j) {
    if (i = 0)          return j;
    else if (j = 0)     return i;
    else
        return min (   d(i-1,j) + 1,
                       d(i,j-1) + 1,
                       d(i-1,j-1) + t(A[i],B[j]));
}
function t(c1, c2) {
    if (c1 = c2)     return 0;
    else                return 1;
}
```

- Komplexität?
 - Für n,m erfolgen 3 Aufrufe, die wiederum jeweils 3 Aufrufe auslösen, die ...
 - Komplexität damit mindestens $O(3^{\min(n,m)})$

Sicher nicht optimal

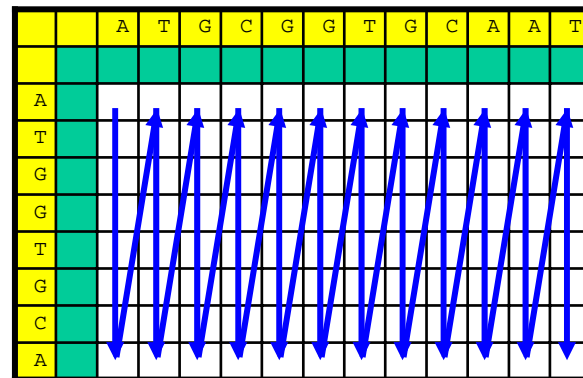
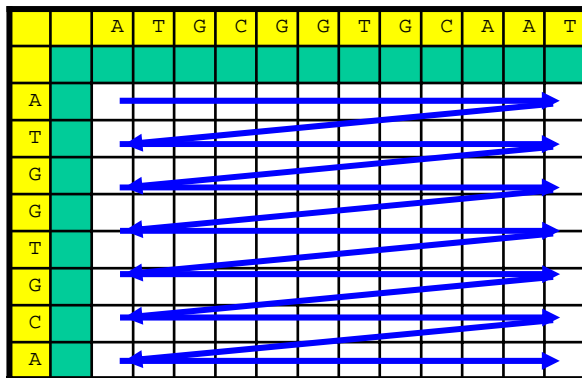
- Durch die Rekursionsgleichung werden viele Teillösungen mehrfach berechnet



- Es gibt nur $(n+1) * (m+1)$ verschiedene Aufrufe
- Wie kann man die redundanten Berechnungen sparen?

Tabellarische Berechnung

- Grundidee
 - Speichern der Teillösungen in Tabelle
 - Bei Berechnung Wiederverwendung wo immer möglich
- Aufbau der Tabelle: Bottom-Up (statt rekursiv Top-Down)
 - **Initialisierung** mit festen Werten $d(i,0)$ und $d(0,j)$
 - **Sukzessive Berechnung** von $d(i,j)$ mit steigendem i,j
 - Für $d(i,j)$ brauchen wir $d(i,j-1)$, $d(i-1,j)$ und $d(i-1,j-1)$
 - Verschiedene Reihenfolgen möglich



Beispiel

$$d(i, j) = \min \left\{ \begin{array}{l} d(i, j-1) + 1 \\ d(i-1, j) + 1 \\ d(i-1, j-1) + t(i, j) \end{array} \right\}$$

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1							
T	2							
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0						
T	2							
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2							
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

Was ist gewonnen?

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

- Editabstand von ATGG, ATGCGGT ist 3
- Wir suchen aber ein Alignment, nicht nur den Abstand
- Extraktion aus der Tabelle durch „Traceback“
 - Bei Berechnung von $d(i,j)$ behalte Pointer auf minimale Vorgängerzelle(n)
 - Die muss nicht eindeutig sein

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

Vom Pfad zum Alignment

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

- Jeder Pfad von (n,m) nach $(1,1)$ ist ein optimales Alignment
 - Starte von $(1,1)$
 - Nach rechts: Deletion in A
 - Nach unten: Insertion in A
 - Diagonal: Match/Replace

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

ATGCGGT
ATG_G__

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

ATGCGGT
AT__GG_

Komplexität

- Aufbau der Tabelle
 - Zur Berechnung einer Zelle muss man genau drei andere Zellen betrachten
 - Konstante Zeit pro Betrachtung
 - $m \cdot n$ Zellen
 - Insgesamt: $O(m \cdot n)$
- Traceback
 - Man kann einen beliebigen Pfad wählen
 - Es muss einen Pfad von (n, m) nach $(1, 1)$ geben
 - Jede Zelle hat mindestens einen Pointer
 - Keine Zelle zeigt aus der Tabelle hinaus
 - Worst-Case Pfadlänge ist $O(m+n)$
- Zusammen
 - $O(m \cdot n)$ (für $m \cdot n > m+n$)

Lokales und globales Alignment

- Die bisherigen Methoden berechnen **globale Alignments**
 - Beide Strings werden komplett betrachtet
 - Größe zusammenhängender Matchregionen spielt keine Rolle
- Das entspricht i.d.R. nicht der biologischen Realität
 - Evolution verschiebt Blöcke von Teilsequenzen
 - Blöcke bestimmen Funktion (Gene, Exons, Proteindomänen, ...)
- Suche nach Alignments mit zusammenhängenden Blöcken
 - „Lokale“ **Subalignments** in den zwei Sequenzen

```
ACCCTATCGATAGCTAGGAAGCTCGATAAATACCGACCAGTAT
AGGAGTCGATAAATACATATAAGAGATAGAATATATTGATG
```

```
ACCCTATCGATA--GC-TAGGAAGCTCGATAAATACCGACCAGTAT-
|          | | | | | |          | | | | | |          | | | | | |
A-GGAGTCGATAAATACATATAAG-A-GATAGAATATA-TTG-ATG
```

Lokale Alignments

- Definition. *Gegeben zwei Strings A, B.*

- *Finde Substrings $a \in A, b \in B$ so dass*

$$sim(a, b) = \max_{\forall a' \in A, b' \in B} (sim(a', b'))$$

- *Das vom (globalen) Alignment von a und b induzierte Alignment von A und B heißt **lokales Alignment***

- Bemerkung

- Lokale Alignments sind unempfindlich gegen **unterschiedliche lange Strings**
- Wichtigkeit der „Blockung“ hängt von Scoring Funktion ab

- Beispiel

- Lokales A. findet den identischen Substring
- Das ist die **biologisch wichtige Information**

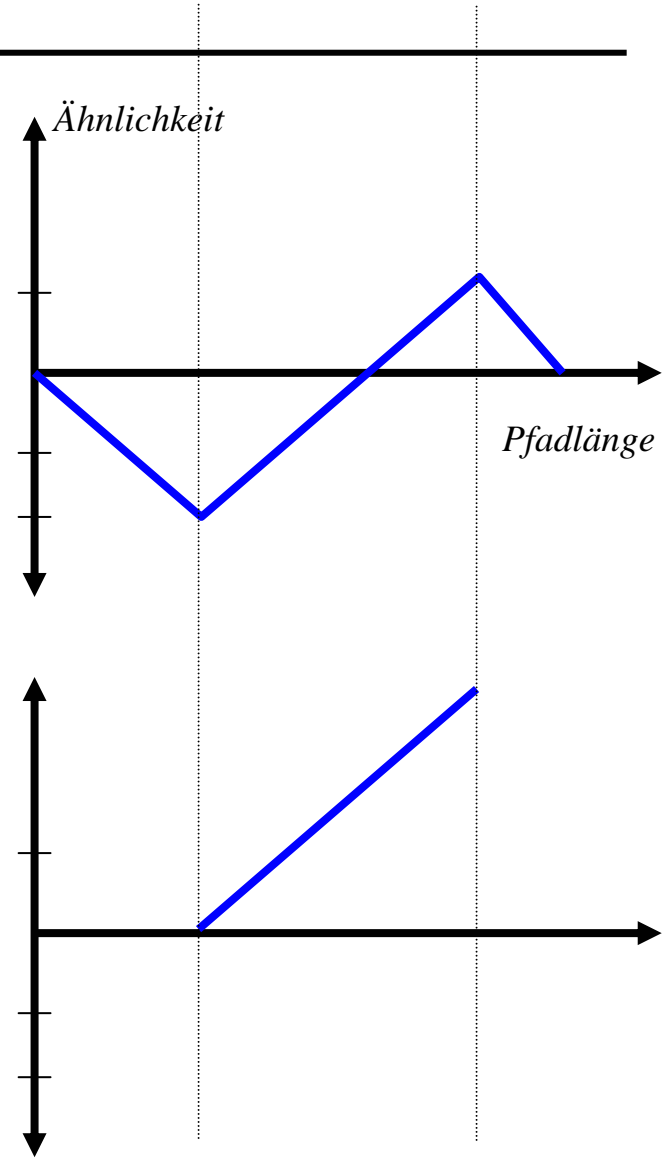
A	G	A	A	G	C	T	C	G	A	T	A	A	T	A	C	C	G	A	C	C	A	G	T	-	A	T	
A	G	G	A	G	-	T	C	G	A	T	A	A	T	A	C	A	T	A	T	A	A	G	A	G	A	T	

Match: +1
I/R/D: -1

Beispiel

		A	T	G	T	G	G
	0	-1	-2	-3	-4	-5	-6
G				-1			
T					0		
G						1	
A							0

		A	T	G	T	G	G
	0	-1	0	-3	-4	-5	-6
G				1			
T					2		
G						3	
A							0



Beispiel 2

Match: +1
I/R/D: -1

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

ATGTCG
ATG____
ATGTCG
AT____G
ATGTCG
A__T_G

➤ Drei Lösungen, alle mit gleicher Güte

		A	T	G	T	C	G
	0	0	0	0	0	0	0
A	0	1	0	0	0	0	0
T	0	0	2	1	1	0	0
G	0	0	1	3	2	1	0

ATGTCG
ATG____

➤ Eine Lösung – das lokale Alignment

Lösen des lokales Suffixalignmentproblems

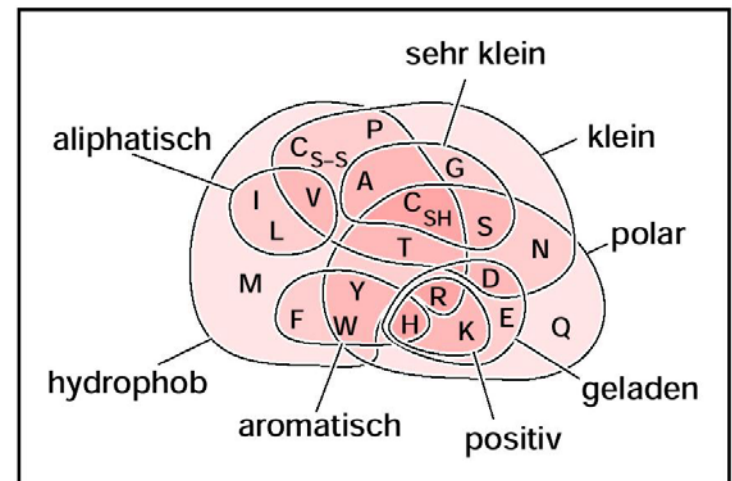
- Theorem.
Gegeben Strings A,B. Dann gilt

$$v(i, j) = \max \left\{ \begin{array}{l} 0 \\ d(i, j-1) + s(_, B[i]) \\ d(i-1, j) + s(A[i], _) \\ d(i-1, j-1) + s(A[i], B[j]) \end{array} \right\}$$

- Beweisidee
 - Beweis läuft sehr ähnlich zum Beweis der ursprünglichen Rekursionsformel
 - Einzige Ausnahme ist die „0“ – der Reset
- Traceback
 - Starte beim **maximalen Wert in der Matrix (nicht notwendigerweise am Rand)**
 - Verfolge beliebigen Pfad bis zu einer **Zelle mit Wert 0**

Hintergrund

- Schon öfters angesprochen ...
 - Ähnlichkeitsmatrizen, Substitutionsmatrizen, Scorefunktionen, ...
- Ersetzung einer Base/Aminosäure durch eine andere hat **unterschiedliche Bedeutung**
 - Basen: Auswirkungen auf kodiertes Protein nicht gleichverteilt über die drei Codon-Positionen
 - Aminosäuren
 - Ersetzung mit „sehr ähnlichen“ Aminosäuren ändert Proteinstruktur kaum
 - Ersetzung mit „wenig ähnlichen“ Aminosäuren kann Struktur vollkommen ändern



Substitutionsmatrizen

- Bewertung **individueller Substitutionen**
- Alignmentalgorithmus ist davon unberührt, nur Parameter ändern sich
- Das Ergebnis kann sich aber **vollkommen ändern**
- Beispiele: Blosom62, Identitätsmatrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	2	2	0	-3	-2	-1	-2	-1	1	-4	-3
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4

	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

Woher nehmen?

- Wie kann man sinnvolle Werte für die Matrix bestimmen?
 - Wir wollen **Ähnlichkeit der biologischen Bedeutung** messen
- Möglichkeit 1: Chemische Eigenschaften
 - Ladung, Größe, Polarität, ...
 - Viele Faktoren mit unklaren Gewichten
 - Wie soll man das durch **ein Bewertungsschema** ausdrücken?
 - Keine Verwendung in der Praxis
- Möglichkeit 2: Beobachtung
 - **Beobachtung der Evolution** statt analytischer Vorhersage
 - Lernen aus Beispielen, also „tatsächlich“ vorgekommener Mutationen
 - Benötigt große Menge homologe Sequenzen