

Bioinformatik

Dynamische Programmierung
Globale und lokale Alignierung



Ulf Leser

Wissensmanagement in der
Bioinformatik



Wie ähnlich sind sich zwei Sequenzen?

- „AGGTAG“ und
 - AGTAG G zu wenig
 - AGGTAG Identisch = überaus ähnlich
 - AGGATAG A zu viel
 - AGTTCAG G durch T ersetzen und C löschen
 - ...TGAGGTAGGTT... **Sehr viel löschen**
- Welche Matches sind besser?
 - „Ähnlichkeit“ muss quantifiziert werden
 - Idee: Wie sehr muss man **eine Sequenz verändern**, um die andere zu erzeugen
 - Man konnte auch Buchstaben zählen, Länge vergleichen, Anzahl GC nehmen, ...

Dotplot

- Definition:

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

- Die Spalten entsprechen den Zeichen von A
- Die Zeilen entsprechen den Zeichen von B
- $M[a,b]=1$ (blau) gdw. $A[a] = B[b]$

- Beispiel

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

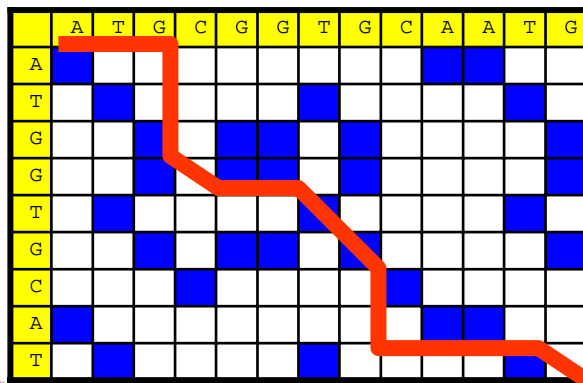
Editabstand

- Offensichtlich gibt es für A,B ziemlich viele Editskripte
- 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
 - | | |
|---------------|---------------|
| I M M M M M D | D M M M M M I |
| _ A G A G A G | A G A G A G _ |
| G A G A G A _ | _ G A G A G A |

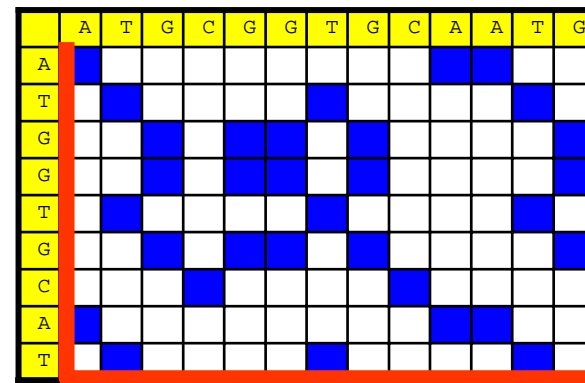
Alignments und Dotplots 2

- Übersetzung von Pfaden im Dotplot in Alignments
 - Dotplot: Sei A horizontal und B vertikal aufgetragen
 - Alignment: Sei A über B angeordnet
 - Schritt nach rechts: Nächstes Zeichen von A; „_“ in B
 - Schritt nach unten: Nächstes Zeichen von B; „_“ in A
 - Schritt nach rechts-unten: Nächstes Zeichen von A und B

ATG__CGGTG__CAATG
 __ATGG__TGCA__T



_____ATGCGGTGCAATG
 ATGGTGCCAT_____



Berechnung von Editabständen

- Definition.

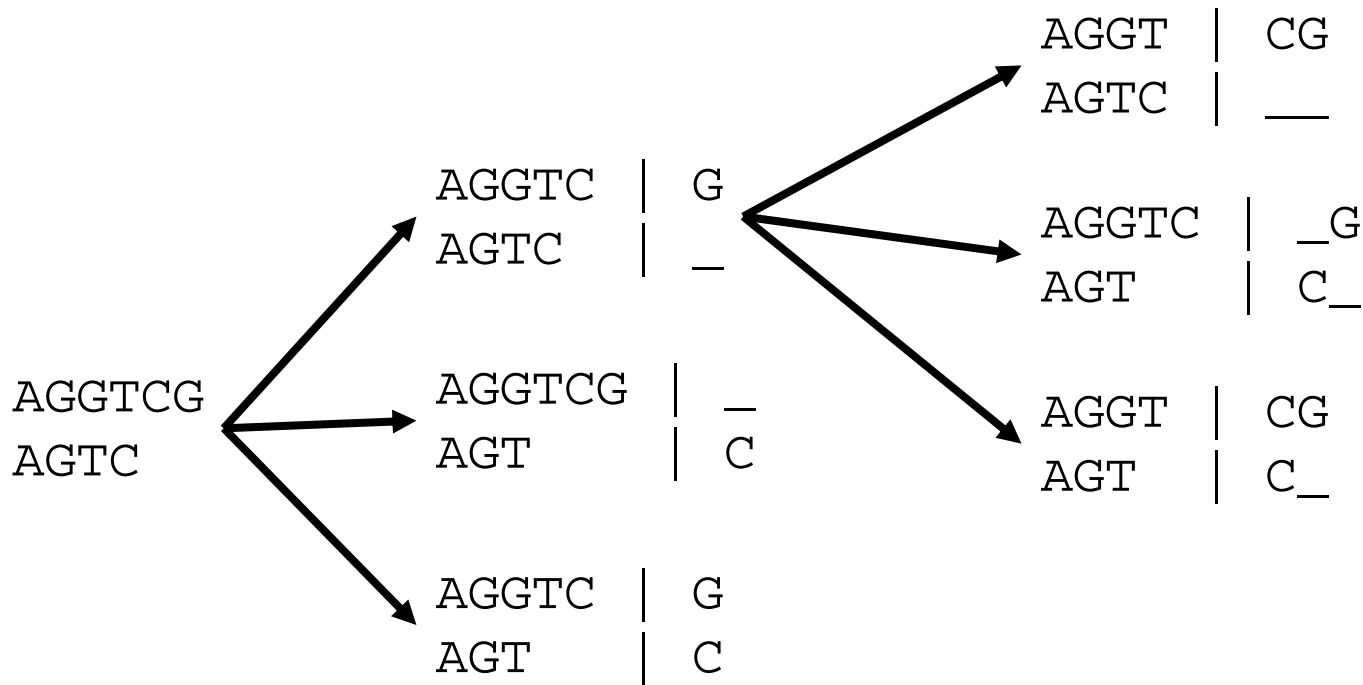
Gegeben zwei Strings A , B mit $|A|=n$, $|B|=m$

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

- Bemerkungen

- Editskript besteht aus Match (M – zählt 0), Einfügungen (I), Löschungen (D) und Ersetzungen (R)
 - Achtung: Jedes R kann durch Kombination $\{I,D\}$ ersetzt werden; im Augenblick werden also R bevorzugt
- Offensichtlich gilt: **$d(n, m) = \text{dist}(A,B)$**
- Definition von $d(i,j)$ dient zur rekursiven Berechnung von $\text{dist}(A,B)$

Rekursive Definition von Alignments



Zusammen

- Theorem

- Der *Editabstand zweier Strings* A, B mit $|A|=n$, $|B|=m$ berechnet sich mit Startbedingung

$$d(i,0) = i \quad d(0, j) = j$$

als $d(n,m)$ mit folgender *Rekursionsgleichung*

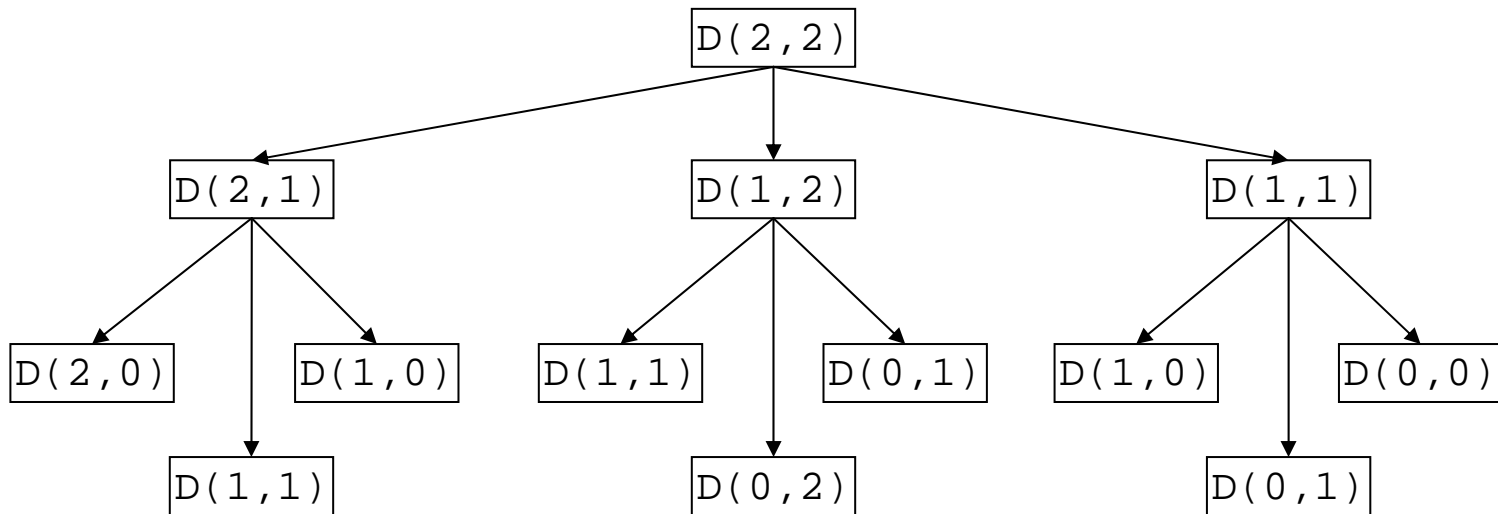
$$d(i, j) = \min \left\{ \begin{array}{l} d(i, j-1) + 1 \\ d(i-1, j) + 1 \\ d(i-1, j-1) + t(i, j) \end{array} \right\}$$

- Beweis

- Per Konstruktion

Sicher nicht optimal

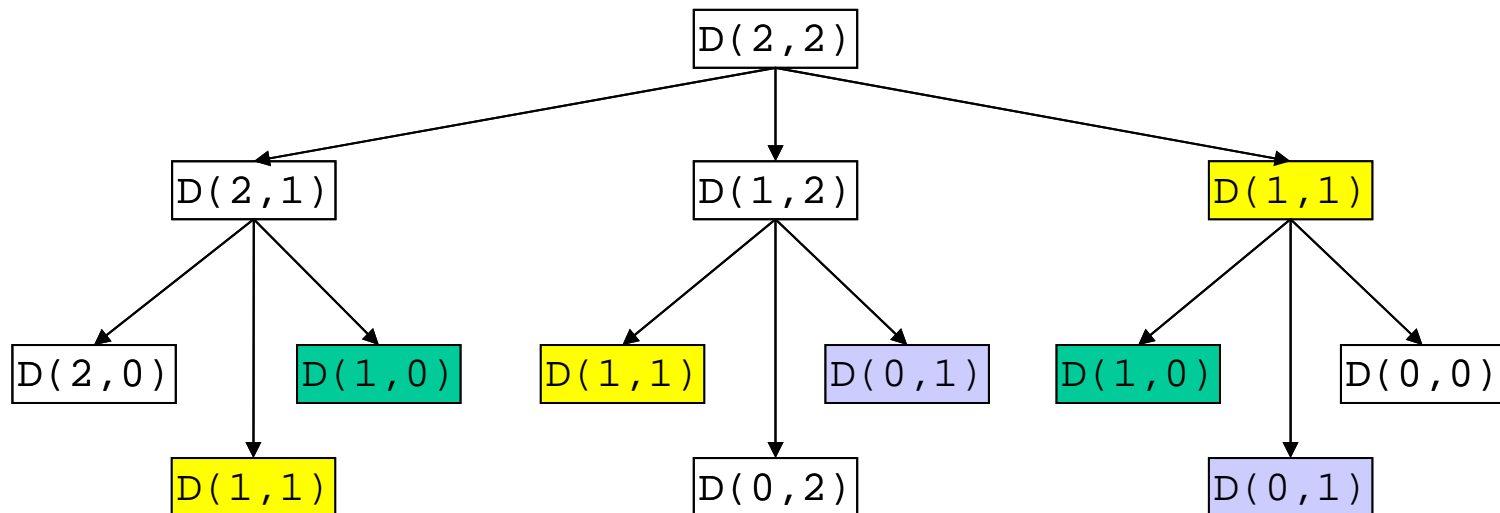
- Aufrufbaum mit Parametern



- Optimierungspotential?

Sicher nicht optimal

- Durch die Rekursionsgleichung werden viele Teillösungen mehrfach berechnet

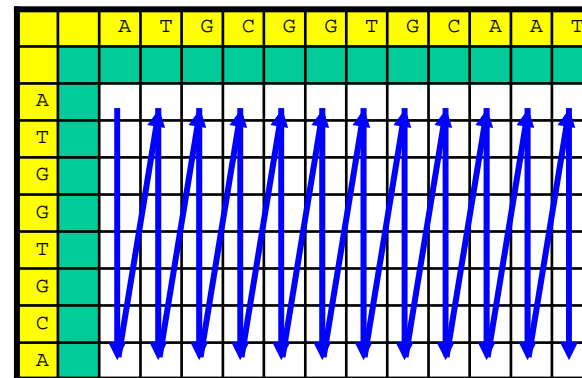
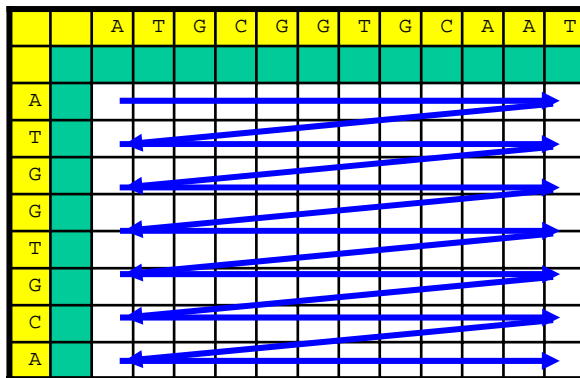


- Es gibt nur $(n+1) * (m+1)$ verschiedene Aufrufe
- Wie kann man die redundanten Berechnungen sparen?

-
- Weiter geht's

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_

Beispiel 2

	0	1	2	3	4	5	6	7
	w	r	i	t	e	r	s	
0	0	1	2	3	4	5	6	7
1	v	1	1	2	3	4	5	6
2	i	2	2	2	2	3	4	5
3	n	3	3	3	3	3	4	5
4	t	4	4	4	4	3	4	5
5	n	5	5	5	5	4	4	5
6	e	6	6	6	6	5	4	5
7	r	7	7	6	7	6	5	4

	0	1	2	3	4	5	6	7
	w	r	i	t	e	r	s	
0	0	1	2	3	4	5	6	7
1	v	1	2	3	4	5	6	7
2	i	2	2	2	3	4	5	7
3	n	3	3	3	3	4	5	6
4	t	4	4	4	4	3	4	5
5	n	5	5	5	5	4	5	6
6	e	6	6	6	6	5	5	6
7	r	7	7	6	7	6	5	4

	0	1	2	3	4	5	6	7
	w	r	i	t	e	r	s	
0	0	1	2	3	4	5	6	7
1	v	1	2	3	4	5	6	7
2	i	2	2	2	3	4	5	7
3	n	3	3	3	3	4	5	6
4	t	4	4	4	4	4	5	6
5	n	5	5	5	5	4	5	6
6	e	6	6	6	6	5	5	6
7	r	7	7	6	7	6	5	4

	0	1	2	3	4	5	6	7
	w	r	i	t	e	r	s	
0	0	1	2	3	4	5	6	7
1	v	1	1	2	3	4	5	6
2	i	2	2	2	3	4	5	7
3	n	3	3	3	3	4	5	6
4	t	4	4	4	4	3	4	5
5	n	5	5	5	5	4	5	6
6	e	6	6	6	6	5	5	6
7	r	7	7	6	7	6	5	4

WRIT_ERS
VINTNER_

WRI_T_ERS
V_INTNER_

WRI_T_ERS
VINTNER

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 $m \cdot n > m+n$)

Prinzip des dynamischen Programmierens

- Ziel: Optimierung einer Zielfunktion
 - Hier: Editabstand
- Prinzip
 - Berechnung von Lösungen für „große“ Problemen aus Lösungen für „kleinere“ Probleme
 - Weiteres Beispiel: Kürzeste Wege in Graphen
- Drei Bestandteile
 - Berechnung aus **optimalen Teillösungen**
 - Hier: Rekursionsgleichung
 - **Zwischenspeichern der Teillösungen**
 - Hier: Tabelle mit Werten $d(i,j)$
 - **Rückverfolgung** der Lösung aus Tabelle
 - Hier: Traceback des Alignments

Needleman-Wunsch Algorithmus

- Historisch der **erste Algorithmus** zum globalen Alignment zweier Sequenzen
 - S.B. Needleman, C.D. Wunsch, „A general method applicable to the search for similarities in the amino acid sequence of two proteins“, J. of Molecular Biology, 48, 1970
- Variante des Algorithmus, wie wir ihn kennen gelernt haben
- Komplexität ist $O(n^3)$ (wenn $m=n$)
- Wird nicht mehr verwendet

-
- Wir können jetzt
 - **Homologie von Sequenzen** bestimmen
 - Die Ähnlichkeit zweier (kompletter) Sequenzen berechnen
 - Genau angeben, welche Veränderungen sich mindestens zugetragen haben müssen
 - Nächste Schritte
 - Lokales Alignment: Ähnlichkeit von Subsequenzen
 - Alignment mit Gaps

Ähnlichkeit

- Welche Frage will man eigentlich beantworten?
 - Wie weit entfernt sind diese beiden Sequenzen
 - Wie ähnlich sind sich zwei Sequenzen
- Ähnlichkeit ist oft einfacher
 - Intuitives Maß – je ähnlicher, desto höher ist die Wahrscheinlichkeit für ähnliche Funktion
 - Programme berechnen i.d.R. Ähnlichkeit
 - **Logisch sind Ähnlichkeit und Abstand äquivalent**
- Differenzierte Betrachtung
 - Ähnlichkeit einzelner Zeichen / Basen / Aminosäuren
 - Ähnliche Zeichen – hohe positive Werte
 - Unähnliche Zeichen – negative Werte

Formal

- Definition

Gegeben Alphabet $\Sigma' = \Sigma \cup _$, Strings A, B über Σ' mit $|A| = |B| = n$

- Eine *Scoringfunktion* ist eine Funktion $s: \Sigma' \times \Sigma' \rightarrow \text{Integer}$
 - Substitutionsmatrix
- Die *Ähnlichkeit* von A, B bzgl. der Scoringfunktion s ist

$$\text{sim}(A, B) = \sum_{i=1}^n s(A[i], B[i])$$

- Bemerkung

- Wir betrachten i.d.R. Alignments, nicht beliebige A, B
- Optimales Alignment ~ Alignment mit höchster Ähnlichkeit

Beispiel

$$\Sigma' = \{A, C, G, T, _ \}$$

	A	C	G	T	_
A	4	-2	-2	-2	0
C		4	-2	-2	-2
G			4	-1	0
T				4	-2
_					0

A	C	_	G	T	C
A	G	G	T	_	C

= 3

A	C	G	T	C
A	G	G	T	C

= 18

Rekursionsgleichung

- Nur kleine Veränderung

$$d(i,0) = \sum_{i=1}^n s(A[i], _) \quad d(0, j) = \sum_{i=1}^m s(_, B[i])$$

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

Lokales und globales Alignment

- Die bisherigen Methoden berechnen **globale Alignments**
 - Beide Strings werden komplett betrachtet
 - Anzahl Matches im Alignment werden maximiert – egal, wo
- 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**

- 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

Smith-Waterman Algorithmus

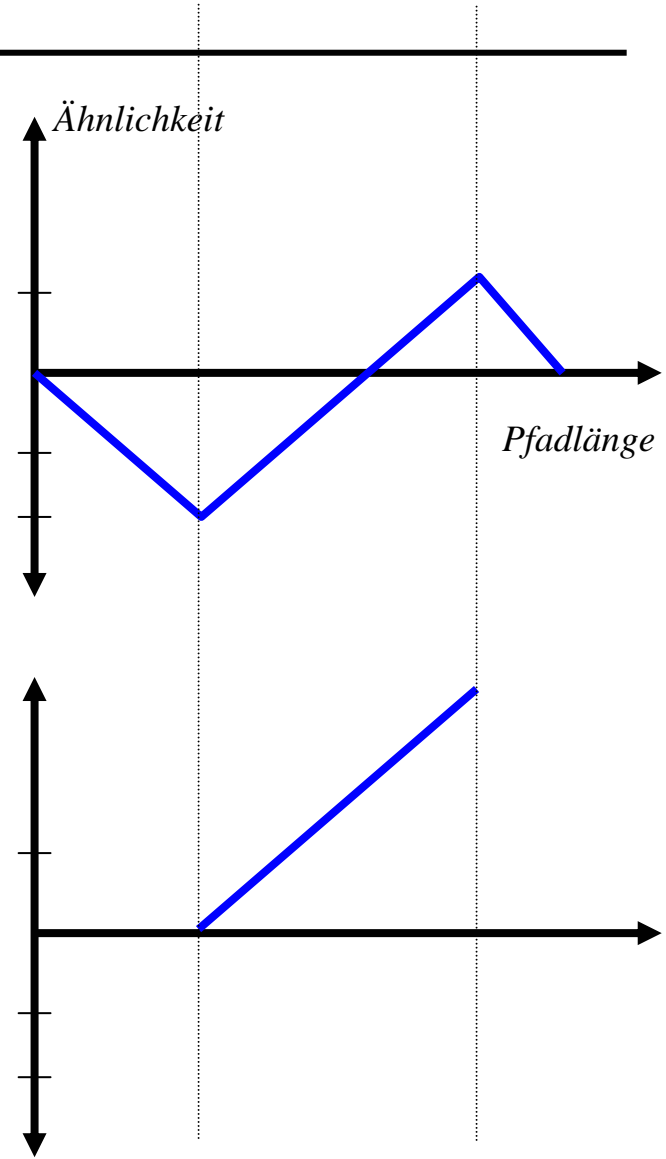
- Smith, Waterman: „Identification of common molecular subsequences“, J. Mol. Bio 147, 1981
- Grundidee
 - Geringfügige Veränderung des Algorithmus für globales Alignment
 - Annahme: Scoring Funktion mit positiven Werten für Matches und negativen Werten für alles andere
 - Eine Reihe von Matches beim Vergleich erzeugt also sukzessive größere Ähnlichkeitswerte
 - Bei Mismatch oder Insertion wird der Wert wieder kleiner
 - Um lange zusammenhängende Regionen zu erhalten, können wir Substringmatches mit negativem Gesamtwert vergessen
 - Also: Statt in negative Werte zu rutschen, darf der Algorithmus auch bei 0 anfangen

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



Optimales lokales Alignment

- Theorem

Gegeben Strings A, B . Sei v^ der Ähnlichkeitswert des besten lokalen Alignments von A, B . Dann gilt mit*

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

– schließlich: $v^* = v(|A|, |B|)$

- Traceback

– Starte beim **maximalen Wert** in der Matrix

– Verfolge beliebigen Pfad bis zu einer **Zelle mit Wert 0**

Beispiel

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

Wann lokales, wann globales Alignment

- Globales Alignment
 - Genauer Vergleich ähnlicher Sequenzen
 - Z.B. Charakterisierung von Protein-Familien
 - Z.B. Bestimmung einer Consensus-Sequenz für MSA
- Lokales Alignment
 - Finden der interessanten Regionen in unbekanntem Sequenzen
 - Z.B. unterschiedliche Spezies, unterschiedliche Gene, ...
 - Finden konservierter (=funktionaler) Subsequenzen
 - Erster Schritt zur Proteinfamilie; danach g. A. innerhalb der Familie
 - Finden konservierter Bereiche zur Untersuchung von Abstammung / evolutionären Prozessen

- Gapped Alignment

- Gap = Loch = Zusammenhänge Folge von Spaces = Insertions bzw. Deletions
- Bisher zählt jedes Space einzeln
- Ist das immer das richtige Modell?

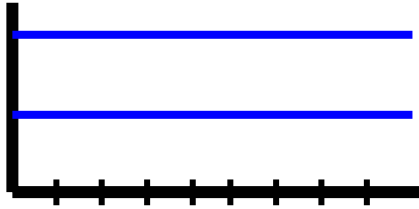
Gaps

- Evolution besteht nicht nur aus Punktmutationen
- Oft werden **ganze Blöcke** verschoben
 - Beispiele
 - Crossing-Over während Meiose (geschlechtliche Zellteilung)
 - „Versehentliche“ Duplikation von Sequenzen
 - Transposable Elements
 - Einbau von Sequenzblöcken an anderer Stelle
 - Im Alignment: lange Reihe von Inserts / Deletions
- Die „guten“ Blöcke sind relevant, die schlechten Zwischenräume nicht
 - **Exons und Introns** unterliegen unterschiedlichem Selektionsdruck
 - **Konservierte Regionen** erscheinen als konstante Blöcke in unterschiedlichen Genomen
 - Proteine setzen sich aus „**active sites**“ und variablen Zwischensequenzen zusammen

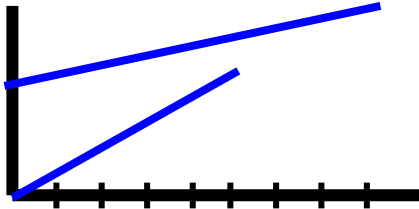
Bewertung von Gaps

- Gesucht: **Gapscorefunktionen**, die Gaps unterschiedlicher Länge flexibel bewerten können
- Sei $w(k)$ der Score eines Gaps der Länge k
- Folgende Funktionsklassen
 - **Konstanter** Gapscore c
 - $w(k) = c$
 - Score unabhängig von Gaplänge
 - **Linearer** Gapscore mit Kosten für Gapbeginn w_s und Gapfortsetzung w_f
 - $w(k) = w_s + w_f * k$
 - Bisheriges Modell nimmt $w_s=0$ und $w_f=1$ an
 - **Konvexer** Gapscore - Score wird nie kleiner mit wachsendem k , aber immer langsamer größer
 - $w(k) = f(k)$; mit $f'(k) > 0$ und $f''(k) \leq 0$
 - Beispiel: $w = \log(k)$
 - **Beliebiger** Gapscore - Score kann wachsen, fallen, oszillieren, ...
 - $w(k) = f(k)$; mit $f(k)$ beliebig

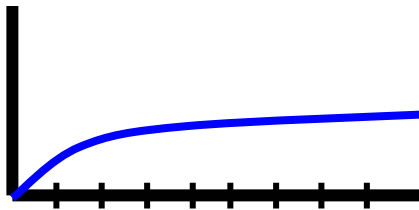
Klassen von Gapscorefunktionen



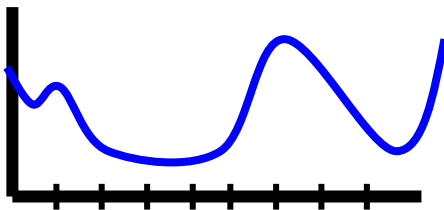
- Konstanter Gapscore



- Linearer Gapscore



- Konvexer Gapscore



- Beliebiger Gapscore

Auswirkungen auf Berechnung

- Je **flexibler die Scorefunktion**, desto komplexer die Berechnung
 - Konstant oder linear: $O(nm)$
 - Konvex: $O(nm \log m)$
 - Beliebig: $O(nm^2 + n^2m)$

Zusammenfassung

- Editabstand und Alignierung ineinander überführbar
- Berechnung eines optimalen Alignments hat **quadratische Komplexität** (wenn $m=n$)
 - Mittels dynamischer Programmierung
 - Tabelle aufbauen, Pfad zurückverfolgen
- Übrigens: **Platzbedarf** ist auch quadratisch
 - Es gibt Erweiterungen mit linearem Platzbedarf
- Varianten
 - Globales versus lokales Alignment
 - Abstand versus Ähnlichkeit
 - Gapscores
 - Es fehlen: **Substitutionsmatrizen**