Algorithmische Bioinformatik

Varianten des Alignmentproblems

Ulf Leser
Wissensmanagement in der Bioinformatik

Ziele

- Varianten des approximativen Stringvergleichs basierend auf Edit-Abstand kennenlernen
- Einsatzmöglichkeiten verstehen

Inhalt dieser Vorlesung

- Gewichtete Editabstände
- Ähnlichkeit
- End-Free Alignment
- Lokales Alignment
- Alignment mit Gaps

Gewichtete Editabstände

- Unsere bisherige Bewertung von evolutionären Ereignissen
  - Matching ist umsonst (keine Evolution)
  - Löschen / Einfügen / Vertauschen ist gleich teuer (1)
- Das muss nicht so sein
  - Aminosäuren sind mehr oder weniger ähnlich
    - Vertauschungen haben unterschiedlich starke Wirkung
    - Ersetzungen (R) sollten unterschiedliche Gewichte haben
  - Lange Gaps wegen Crossing-Over nicht linear schlimmer als kurze
  - Ins/Del auf DNA bewirkt Frameshift – viel schlimmer als Replace
- Man braucht flexiblere Bewertungsschemata

Allgemeines Bewertungsschema

- Operationen werden unterschiedlich bestraft
  - Einfügen: c_i
  - Löschen: c_d
  - Match: m
  - Replace: r(A[i], B[j])
- r(x,y) entnimmt man einer Substitutionsmatrix
  - Beispiel

	A	C	G	T
A	0	1	1	1
C	1	0	.5	1
G	1	.5	0	1
T	1	1	1	0

  - r(x,y) sollte immer kleiner als c_i+c_d sein, da I+D ein R simulieren
  - Nennen wir auch Scoringfunktion; kann Kosten für InDels enthalten

Veränderungen am Algorithmus

- Nur marginale Änderungen

$$d(i,0) = i * c_d \qquad d(0,j) = j * c_i$$

$$d(i,j) = \min \begin{cases} d(i,j-1) + c_i \\ d(i-1,j) + c_d \\ d(i-1,j-1) + r(i,j) \end{cases}$$

- Eigenschaften bleiben erhalten
  - Optimalität, tabellarische Berechnung, Pfade, ...

Beispiel

Scoring r=2, i/d=3

		0	1	2	3	4	5	6	7
			w	r	i	t	e	r	s
0		0	3	6	9	12	15	18	21
1	v	3	2	5	8	11	14	17	20
2	i	6	5	4	5	8	11	14	17
3	n	9	8	7	6	7	10	13	16
4	t	12	11	10	9	6	9	12	15
5	n	15	14	13	12	9	8	11	14
6	e	18	17	16	15	12	9	10	13
7	r	21	20	17	18	15	12	9	12

```
VINTNER_    VINTNER
WRIT_ERS    WRITERS
```

Scoring r=1, i/d=1

		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	7
2	i	2	2	2	2	3	4	5	6
3	n	3	3	3	3	3	4	5	6
4	t	4	4	4	4	3	4	5	6
5	n	5	5	5	5	4	4	5	6
6	e	6	6	6	6	5	4	5	6
7	r	7	7	6	7	6	5	4	5

```
_VINTNER_    V_INTNER_
WRI_T_ERS    WRI_T_ERS
```

```
VINTNER_
WRIT_ERS
```

Inhalt dieser Vorlesung

- Gewichtete Editabstände
- Ähnlichkeit
- End-Free Alignment
- Lokales Alignment
- Alignment mit Gaps

Ähnlichkeit

- Ähnlichkeit und Abstand
  - Je ähnlicher zwei Strings, desto geringer ist der Abstand
  - Man maximiert Ähnlichkeit und minimiert Abstand
  - Logisch ist Ähnlichkeit und Abstand äquivalent
- Ähnlichkeit auf Zeichenebene
  - Ähnliche (gleiche) Zeichen – positive Werte (belohnen)
  - Unähnliche Zeichen – negative Werte (bestrafen)
  - INDEL – negative Werte (bestrafen)

Formal

- Definition
  Geg.: Alphabet Σ‘=Σ∪‘_‘ ; Strings A,B, Scoringfunktion s: Σ‘xΣ‘ → Integer
  - *Die Ähnlichkeit sim(A, B) eines Alignments von A und B bzgl. s ist*

$$sim(A,B) = \sum_{i=1}^{m} s\left(A[i], B[i]\right)$$

    - m: Anzahl Spalten des Alignments
- Bemerkung
  - Bei Ähnlichkeit sind große Werte gut
  - Bei Abstand waren kleine Werte gut (Alignmentabstand)

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
_					

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

= 3

A	C	G	T	C
A	G	G	T	C

= 14

Rekursionsgleichung

- Wieder nur kleine Veränderung
  - Welche?

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

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

Inhalt dieser Vorlesung

- Gewichtete Editabstände
- Ähnlichkeit
- End-Free Alignment
- Lokales Alignment
- Alignment mit Gaps

Spezialfall Assembly

- Assembly braucht keine globalen Alignments

```
ACGTGAT_AGCTAGCTAG----------
----GATCGCT_TGCAAGTATCTCTATAT

                  CAGGTTAGATGACCAGTA_GAAGTGGCA
                  -------GATTACCAGTAGGA------
```

- Welches Problem müssen wir lösen?

End-Free Ähnlichkeit

- Insdels am Anfang und Ende sind umsonst
  - Wie drückt sich das in der Matrix aus?

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

		T	T	G	G
T					
T					
G					

End-Free Ähnlichkeit

- Insdels am Anfang und Ende sind umsonst
  - Anfang: d(i,0) = 0, d(0,j) = 0
  - Wo ist der optimale „end-free“ Pfad?

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

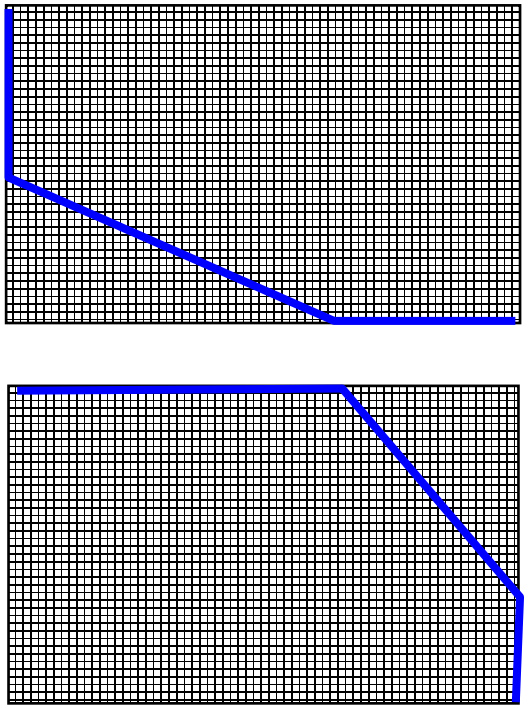
		T	T	G	G
	0	0	0	0	0
T	0	4	4	2	0
T	0	8	8	6	4
G	0	6	6	12	10

TTGG
TT_G

TTGG
TTG_

- End-Free Ähnlichkeit ist das Maximum der Randzahlen
  - Von dort bis (n,m) folgen nur INDEL – am Ende umsonst
  - Die (kostenlosen) Leerzeichen am oberen/linken Rand sind durch die geänderte Initialisierung berücksichtigt

End-Free Pfade



umsonst

_____AAAA_AAAAAAAAAA_AAAAAAAAAAA
BBBBBBBB__BBBB_BBBB_BBBBBBB_____

AAAAAAAAAAAAAAA_AA_AAAAAA_____
_____BBBBBBBB__BBBB_BBBB

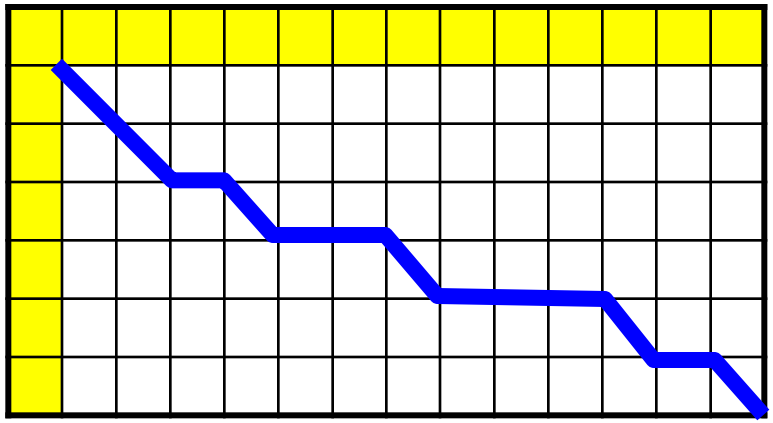
? **AAAAAAAAAAAAAAA_AAAAA_AAAAAAAAAA**
_____BBBBBBBB__BBBB_____

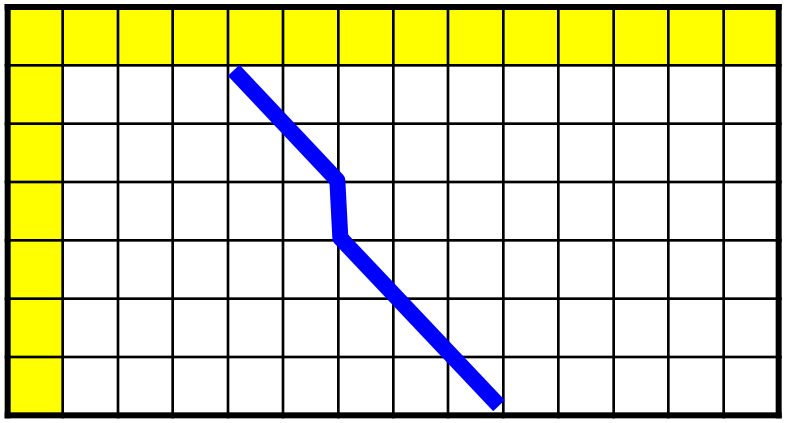
	A	C	G	T	_
A	4	-2	-2	-2	-2
C		4	-2	-2	-2
G			4	-2	-2
T				4	-2
_					0

Beispiel

AGTCAGTGCGTGC
AG_C__T___T_C

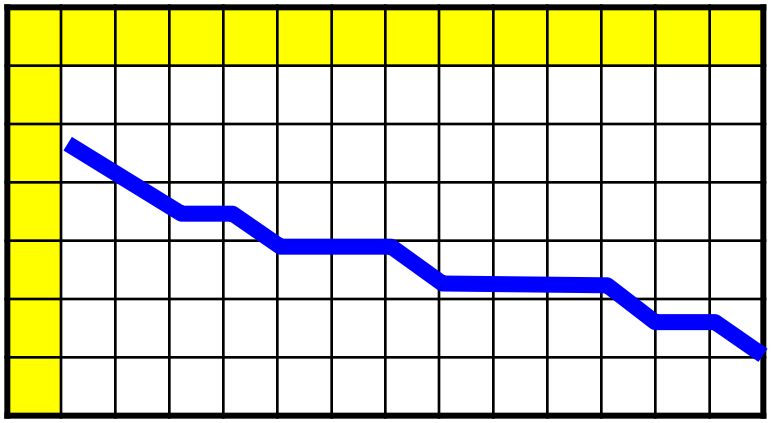
AGTCAG_TGCGTGC
_____AGCTTC_____



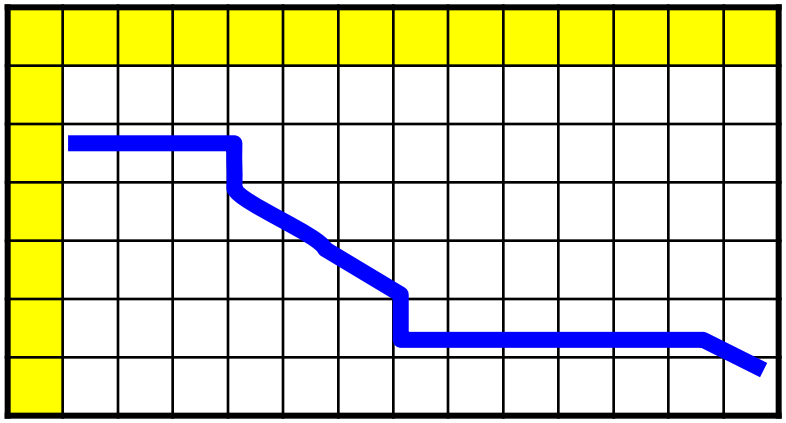


Anderes Beispiel

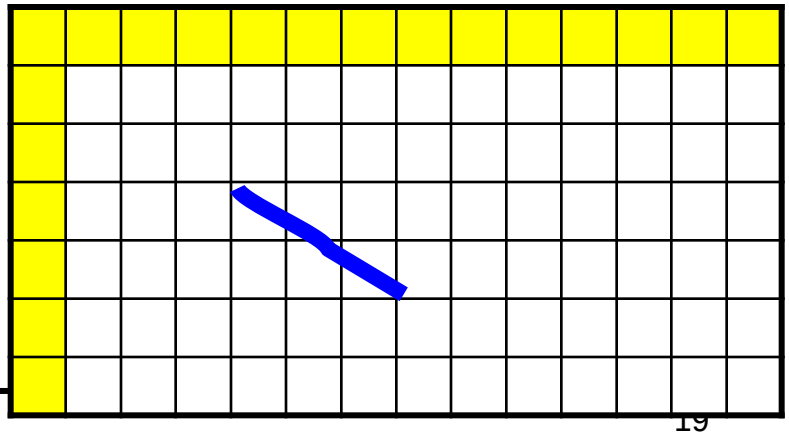
_____AGTCAGTGCGTGC____
TTAAAG_C__T____T_CAAA



____AGTCAG_TGCGTGC____
TTAA____AGCTTC_____CAAA



____AGTCAG_TGCGTGC____
TTAA____AGCTTC_____CAAA



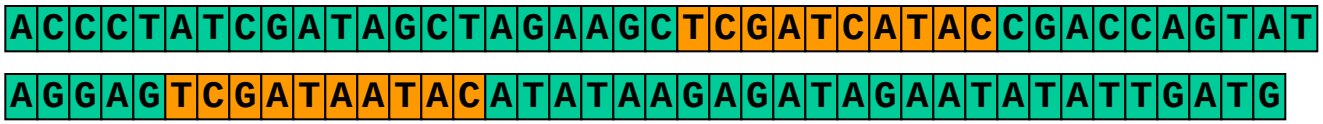
Inhalt dieser Vorlesung

- End-Free Alignment
- Gewichtete Editabstände
- Ähnlichkeit
- Lokales Alignment
- Alignment mit Gaps

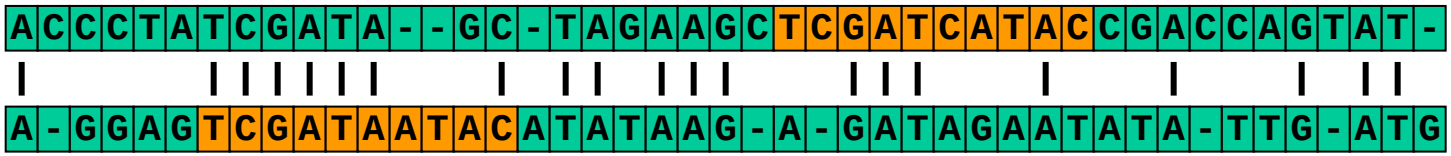
Lokales und globales Alignment

- Globales Alignments betrachtet beide Sequenzen komplett
  - Größe zusammenhängender Matches spielt keine Rolle
- Biologie benötigt oft ein anders Maß
  - Evolution verschiebt Teilsequenzen (Blöcke)
  - Blöcke bestimmen Funktion: Gene, Exons, Domänen, ...
- Also: Suche nach gut passenden Substrings
  - Substrings beider Strings, die sich sehr ähnlich sind

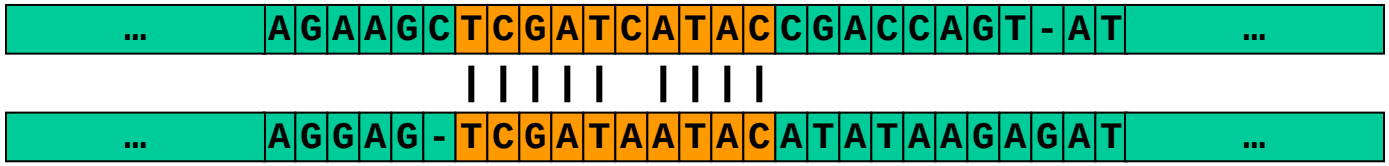
Beispiel



Globales Alignment



Biologisch interessanter



Lokales Alignment

- Definition. *Gegeben zwei Strings A, B.*
  - *Der lokale Ähnlichkeitsscore sim* für A und B ist*

$$sim^*(A,B) = \max_{\forall a=A[x..i], b=B[y..j]}\left(sim(a,b)\right)$$

  - *Das Alignment von a und b heißt lokales Alignment*
  - Bereiche außerhalb von a in A und von b in B werden ignoriert
- Bemerkung
  - Unempfindlich gegen unterschiedlich lange Sequenzen
  - Längere sehr ähnliche Bereiche erreichen bessere Scores als kürzere (aber hätten den gleichen Abstand)
  - Wichtigkeit der „Blockung“ hängt von Scoringfunktion ab

Berechnung lokaler Alignments

- Naiver Ansatz
  - Aufzählen aller Substrings von A und in B
  - Berechnung des optimalen Alignments aller Paare
  - Auswahl des Paares mit der größten Ähnlichkeit
- Komplexität?
  - Sei $|A|=|B|=n$
  - Jeweils $O(n^2)$ Substrings
  - Damit $O(n^4)$ Paare
  - Alignment ist $O(n^2)$
  - Zusammen: $O(n^6)$

Länge des Substrings	n	n-1	n-2	...	1
Anzahl Substrings	1	2	3	...	n

Smith-Waterman Algorithmus

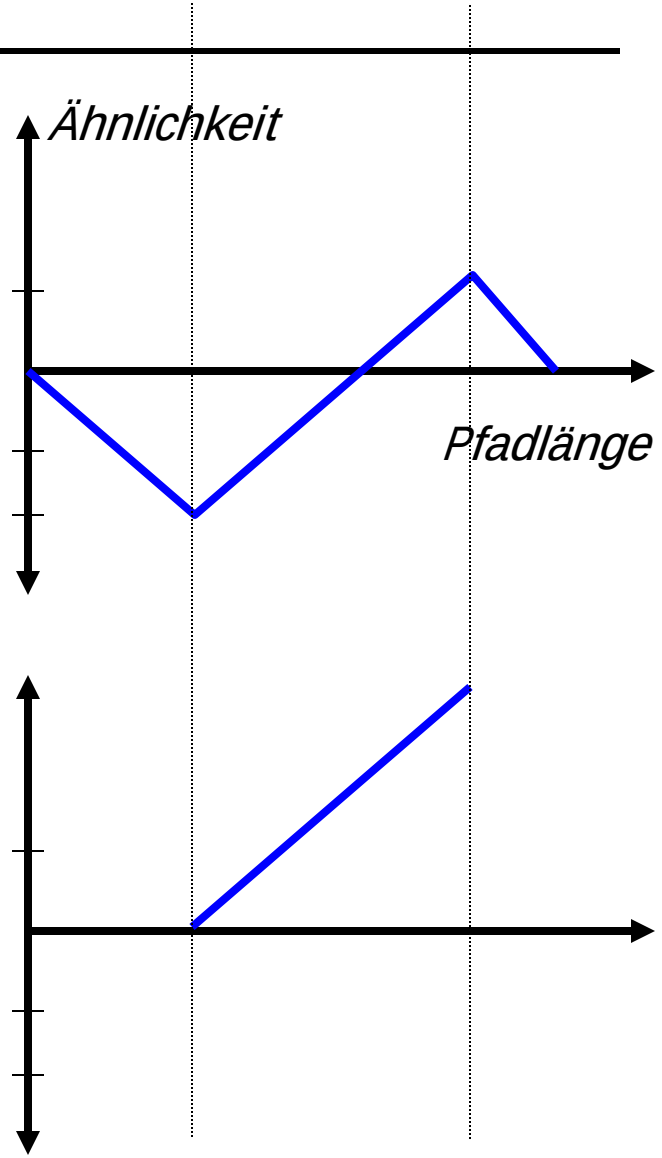
- Smith, Waterman: „Identification of common molecular subsequences“, J. Mol. Bio 147, 1981
- Annahme
  - Wir suchen das Paar von Substrings mit maximaler Ähnlichkeit
    - Editabstand nicht hilfreich – einfach leere Substrings nehmen
  - Scoring: Positive Werte für Matches, negative für Mismatches
- Grundidee
  - Betrachten wir einen beliebigen Pfad durch die Matrix
  - Eine Reihe von Matches erzeugt sukzessiv größere Scores
  - Bei Mismatch oder Insertion wird der Pfadscore wieder kleiner
  - So lange ein positiver Score entlang des Pfades übrig bleibt, kann noch was „Gutes“ entstehen
  - Pfade mit negativem Score kann man sofort vergessen

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				
G				1			
T					2		
G						3	
A							…



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

Voraussetzung

- Definition
  Gegeben Strings A, B und i<n, j<m. Sei a=A[1..i] und b=B[1..j].
  - *Das lokale Suffixalignmentproblem sucht die Suffixe a‘ von a und b‘ von b so, dass sim(a‘,b‘) maximal ist*
  - *Den maximalen Suffixalignmentscore bezeichnen wir als v(i,j) mit*

$$v(i,j) = \max_{\forall a'=A[x..i], b'=B[y..j]} \left(sim(a', b') \right)$$

- Bemerkung
  - Da sim(∅, ∅)=0, ist v(i,j) ≥ 0 ∀ i,j

Folgerung

- Theorem
  Gegeben Strings A, B. Der lokale Ähnlichkeitsscore sim für zwei Zeichenketten A, B berechnet sich als:*

$$sim^*(A,B) = \max_{i \le n,\, j \le m}\left(v(i,j)\right)$$

- Beweis
  - Offensichtlich – alle Paare von Teilstrings werden berücksichtigt
- Folgerung
  - Wenn wir das lokale Suffixalignmentproblem lösen können, können wir auch das lokale Ähnlichkeitsproblem lösen

Lösen des lokales Suffixalignmentproblems

- Theorem
  Gegeben Strings A,B. Dann gilt

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

- Traceback
  - sim* ist der maximale Score aller Suffixalignments
  - Also: Starte beim maximalen Wert in der Matrix
    - Nicht notwendigerweise am Rand
  - Verfolge beliebigen Pfad bis zu einer Zelle mit Wert 0

Wann lokales, wann globales Alignment

- Globales Alignment
  - Erfasst beide Sequenzen komplett
  - Genauer Vergleich ähnlicher Sequenzen
    - Charakterisierung von Protein-Familien
    - Bestimmung einer Consensus-Sequenz für MSA
- Lokales Alignment
  - Erfasst nur Teile der beiden Sequenzen
  - Finden der interessanten Regionen in unbekannten Sequenzen
    - Exons in genomischer Sequenz
    - Domänen in Proteinsequenzen
    - Allgemein: konservierte (=funktionale) Subsequenzen

- Gewichtete Editabstände
- Ähnlichkeit
- End-Free Alignment
- Lokales Alignment
- Alignment mit Gaps
  - Gaps
  - Beliebige Gapkosten
  - Lineare Gapkosten

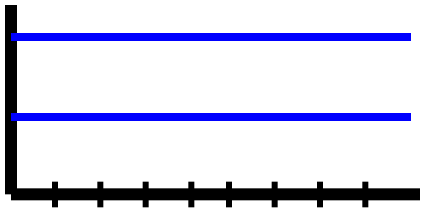
Gaps

- Gap = Zusammenhänge Folge von Leerzeichen
  - Bisher zählt jedes Leerzeichen einzeln
  - Mehr Leerzeichen – linear mehr Kosten
- Evolution: Mehr als Punktmutationen und Insdels
  - Crossing-Over während Meiose (geschlechtliche Zellteilung)
  - „Versehentliche“ Duplikation von Sequenzen
  - Transposable Elements
  - ➢ Erscheinen als Ketten von Insdel
  - ➢ Ist aber nur ein evolutionäres Ereignis
  - ➢ Häufigkeit ist vermutlich nicht abhängig von der Länge des Gaps

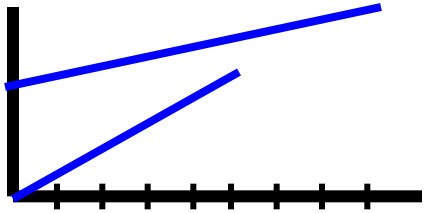
Bewertung von Gaps

- Sei w(k) der Score eines Gaps der Länge k
- Funktionsklassen
  - Konstanter Gapscore c
    - w(k) = c
    - Score unabhängig von Gaplänge
  - Linear 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
  - Konvex: Score wird nie kleiner, 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

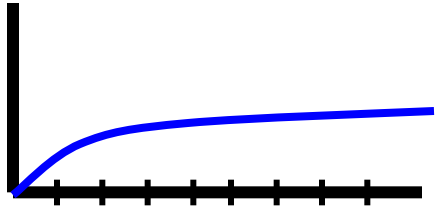
Klassen von Gapscorefunktionen



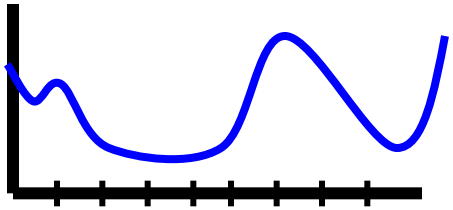
- Konstanter Gapscore



- Linearer Gapscore



- Konvexer Gapscore



- Beliebiger Gapscore

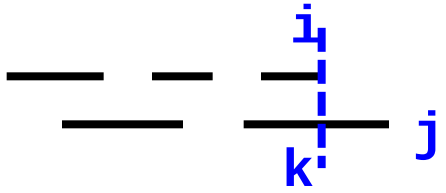
Auswirkungen auf Berechnung

- Je flexibler die Gapscorefunktion, desto komplexer die Berechnung
  - Konstant oder linear: O(n*m)
  - Konvex: O(n*m*log(m))
  - Beliebig: $O(n*m^2 + n^2*m)$
- Wir analysieren lineare und beliebige Gapscorefunktionen
  - Ähnlichkeit - Gapscores gehen also negativ ein
  - Notwendig: Erweiterung der Rekursionsgleichung
- Zunächst: Beliebige Gapscores

Erinnerung – Normalfall, Insertion in A

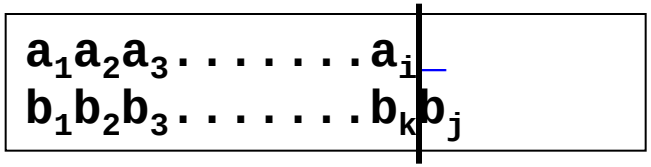
- Fallunterscheidung für die Berechnung von d(i,j)
  - Insertion in A (oder Deletion in B)
    - Situation:
      ```
      ...I
      XXX_
      XXXT
      ```
    - d(i,j) = d(i, j-1) – 1
- Voraussetzung war, dass wir das optimale Alignment von A[1..i] und B[1..j-1] schon kennen
  - Durch das Einfügen eines „_“ wird der Score um 1 schlechter – egal, wie das Alignment vor der aktuellen Position aussieht
  - Für beliebige Gapscores ist das „davor“ aber wichtig – Score ist unterschiedlich, je nachdem wie groß das Gap ist

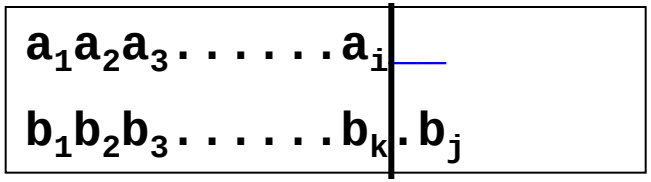
Beliebige Gapscores, Insertion in A



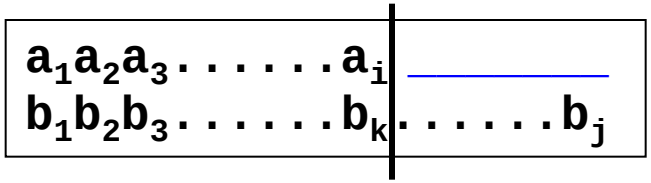
- Wir suchen das optimale Alignment von A[1..i] und B[1..j]
- Nehmen wir an, dass die letzte Spalte „_“ und B[j] ist
  - Das letzte Nicht-Blank von A aligniert mit irgendeinem Zeichen B[k] links von j-1 in B
  - Danach sind in A j-k Blanks und in B j-k Nicht-Blanks – das Gap
  - k steht aber nicht fest – wir suchen den optimalen Fall
  - Wir müssen alle möglichen k betrachten = alle möglichen Längen des letzten Gaps in A vor i

Mögliche Alignments





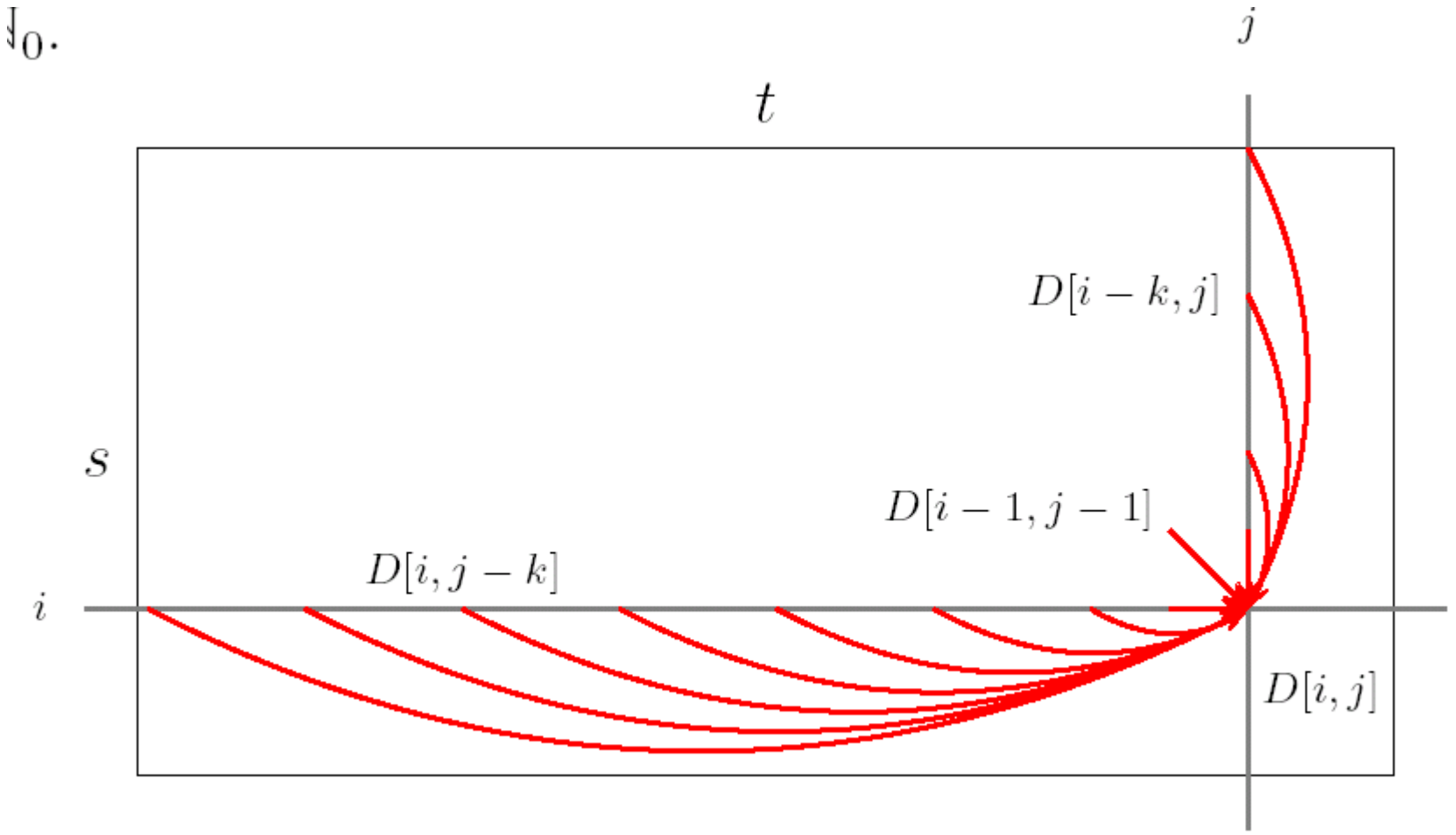
Allgemeiner Fall



- Die optimale Ähnlichkeit für das Alignment bis i,j unter der Annahme, dass die letzte Spalte im Alignment ein Space in A enthält, berechnet sich als:

$$E(i,j) = \max_{1 \le k \le j-1} \left(V(i,k) - w(j-k) \right)$$

Was müssen wir betrachten?



Beispiel

k	w(k)
1	-1
2	-1
3	0
4	-2
…	…

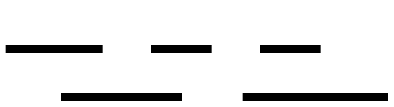
	1	2	3	4
1			2	
2			5	
3			3	
4			4	
5				

- Berechnung von E(3,4)
  - max(3-1, 5-1, 2-0) = 4

	1	2	3	4
1			2	
2			5	
3			3	
4			4	
5			5	

- Berechnung von E(3,5)
  - max(4-1, 3-1, 5-0, 2-2) = 5
  - Betrachtung von E(3,4) reicht nicht!

Rekursionsgleichungen



$$E(i,j) = \max_{1 \le k \le j-1} \left(V(i,k) - w(j-k) \right)$$

$$F(i,j) = \max_{1 \le l \le i-1} \left(V(l,j) - w(i-l) \right)$$

$$G(i,j) = V(i-1, j-1) + s(A[i], B[j])$$

Zusammen: $V(i,j) = \max(E(i,j), F(i,j), G(i,j))$

Komplexität

- Theorem
  Gegeben Strings A,B mit |A|=n, |B|=m. Für beliebige Gapscorefunktionen kann das optimale Alignment in $O(n^2m+nm^2)$ berechnet werden
- Beweis
  - Wir berechnen Einträge einer Tabelle der Größe n*m
  - In jeder Zelle berechnen wir 4 Werte: E(i,j), F(i,j), G(i,j), V(i,j)
    - Berechnung von G und V ist O(1)
    - Für E und F müssen die Zellen (i,j-1), (i,j-2),...,(i,1) und (i-1,j), (i-2,j), ..., (1,j) betrachtet werden
      - Für fixes i (eine Spalte füllen) betrachten wir Σj für $1 \leq j \leq n-1$ Zellen. Das ist $O(n^2)$
      - Für fixe j (Zeile füllen) ist das $O(m^2)$
  - Daraus folgt: $O(n*m^2 + m*n^2)$

Aber ...

- Wir haben den optimalen Wert in [x,y], wobei dafür ein Gap der Länge k, das bei y endet, optimal war
- Nun sind wir bei [x,j>y] und suchen alle Vorgänger ab
  - Wir betrachten auch [x,y] und verlängern das Gap zwischen y und j. Dafür nehmen wir Gapkosten w(j-y) an
- In Wahrheit hätten wir in dem Fall aber doch Gapkosten w(j-y+k)
  - Bei beliebigen Gapkosten gilt nicht: w(j-y-2)=w(2)+w(j-y)
- Machen wir was falsch?

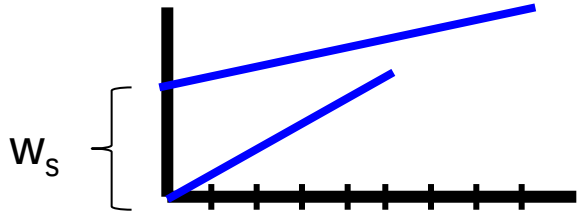
Ansichtssache

- Wenn kurze Gaps billiger sind, dann ...
  - ... drückt das aus, dass kurze Gaps häufigere Ereignisse sind
  - ... sind wahrscheinlich eher zwei Ereignisse mit kurzen Gaps direkt hintereinander als eines mit einem langen Gap passiert
- Kann man diskutieren
  - Unser Alignment enthält nur ein Gap, aber wir zählen zwei kürzere
  - Der Algorithmus findet optimale Stückelungen längerer Gaps
  - Richtig oder falsch?
- Kann man ausschließen, wenn man pro Zelle getrennt optimale Scores für das Erreichen der Zelle mit Gap und ohne Gap am Ende merkt

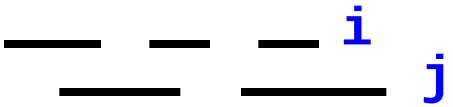
- Gewichtete Editabstände
- Ähnlichkeit
- End-Free Alignment
- Lokales Alignment
- Alignment mit Gaps
  - Gaps
  - Beliebige Gapkosten
  - Lineare Gapkosten

Lineare Gapscorefunktionen

- „Normales“ Alignment: Lineare Gapkosten mit $w_s=0$
- Was muss man tun, wenn $w_s \neq 0$?
- Zwei Möglichkeiten bei Insertion in letzter Spalte
  - Neues Gap: Kostet $w_s + w_f$ (der Fall ist neu)
  - Gap wird fortgesetzt: Kostet w_f (das übliche)



Zwei Fälle, wenn letzte Spalte Insertion ist



- Gap beginnt nach A[i]: $E(i, j) = V(i, j-1) - w_f - w_s$
- Gap in A wird fortgesetzt: $E(i, j) = E(i, j-1) - w_f$
- Zusammen

$$E(i, j) = \max\left(V(i, j-1) - w_f - w_s, E(i, j-1) - w_f\right)$$

Unterschied

$$E(i, j) = \max\left(V(i, j-1) - w_f - w_s, E(i, j-1) - w_f\right)$$

- Vorsicht
  - E/F sind die Kosten bis zu einer Zelle, wenn als letztes eine Insertion/Deletion ausgeführt wurde
  - V sind die optimalen Kosten bis zu einer Zelle, unabhängig davon, welche Operation als letztes ausgeführt wurde
- Wir müssen uns in jeder Zelle E, F und V merken
  - Bei beliebigen Gapscores ist das nicht unbedingt notwendig
    - Aber siehe Folie „Aber...“
- Platzbedarf steigt (um konstanten Faktor)

Rekursionsgleichungen

$$E(i, j) = \max\left(V(i, j-1) - w_f - w_s, E(i, j-1) - w_f\right)$$

$$F(i, j) = \max\left(V(i-1, j) - w_f - w_s, F(i-1, j) - w_f\right)$$

$$G(i, j) = V(i-1, j-1) + s(A[i], B[j])$$

$$V(i, j) = \max(E(i, j), F(i, j), G(i, j))$$

Komplexität

- Theorem
  *Gegeben Strings A,B mit |A|=n, |B|=m. Für lineare Gapscorefunktionen kann das optimale Alignment in O(n*m) berechnet werden*
- Beweis
  - Wir berechnen in jeder Zelle die Werte E(i,j), F(i,j), G(i,j)
  - Für die Berechnung jeder Zelle müssen nur konstant viele andere Zellen betrachtet werden
  - Daraus folgt: O(n*m)
  - qed.

Zusammenfassung

- Alignment ist vielschichtig
  - End-Free versus lokales versus globales Alignment
  - Gewichtete Editabstände (Match/Mismatch/Insert/Del)
  - Gap-Score Modelle (konstant/linear/konvex/beliebig)
- Zusammenspiel ist kompliziert
  - Globales Alignment mit hohen Gapkosten erzeugt sehr kompakte Alignments (viele Mismatches)
  - Lokales Alignment mit niedrigen Gapkosten ähnelt irgendwann globalem Alignment
- Wahl eines Modells abhängig von der konkreten biologischen Fragestellung

Selbsttest

- Erklären Sie den Unterschied zwischen globalem und lokalem Alignment
- Nehmen Sie an, alle Reads während einer Sequenzierung sind gleich lang. Welche Art Alignments müsste man ausrechnen? Ist das End-Free Alignement?
- Wann benutzt man eher globales, wann eher lokales Alignment?
- Beweis der Komplexität bei beliebigen Gap-Scores
- Welche Komplexität hat globales Alignment mit konstanten Gap-Scores? Warum?