

Bioinformatik

Alignment mit linearem Platzbedarf
K-Band Alignment



Ulf Leser
Wissensmanagement in der
Bioinformatik



Spezialfall Assembly

- Assembly beim Shotgun-Sequenzieren braucht keine Alignments der kompletten Strings

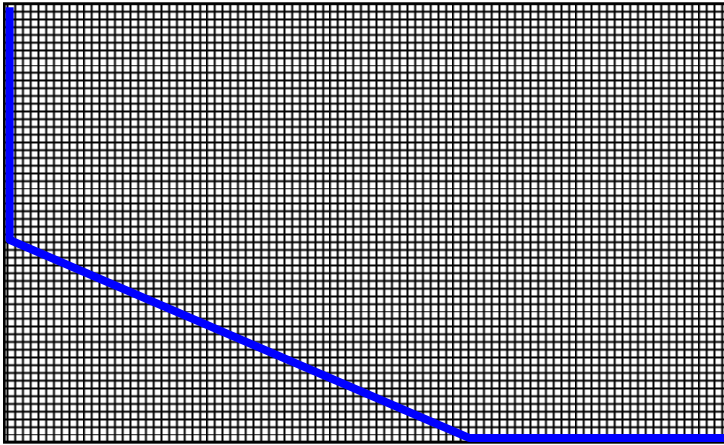


```
ACGTGATAGCTAGCTAG-----  
-----GATCGCTTGCAAGTATCTCTATAT
```

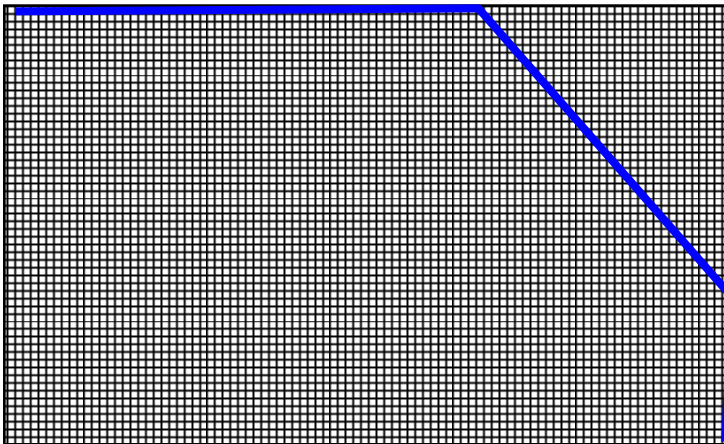
```
CAGGTTAGATGACCAGTAGCAGTGGCA  
-----GATTACCAGTAGGA-----
```

- Modellierung?
 - Spaces am **Anfang / Ende** des Alignments sind umsonst

End-Free Pfade



_____ AAAA_ AAAAAAAAAA_ AAAAAAAAAA
BBBBBBBBB_ BBBB_ BBBB_ BBBBBBB_ _____



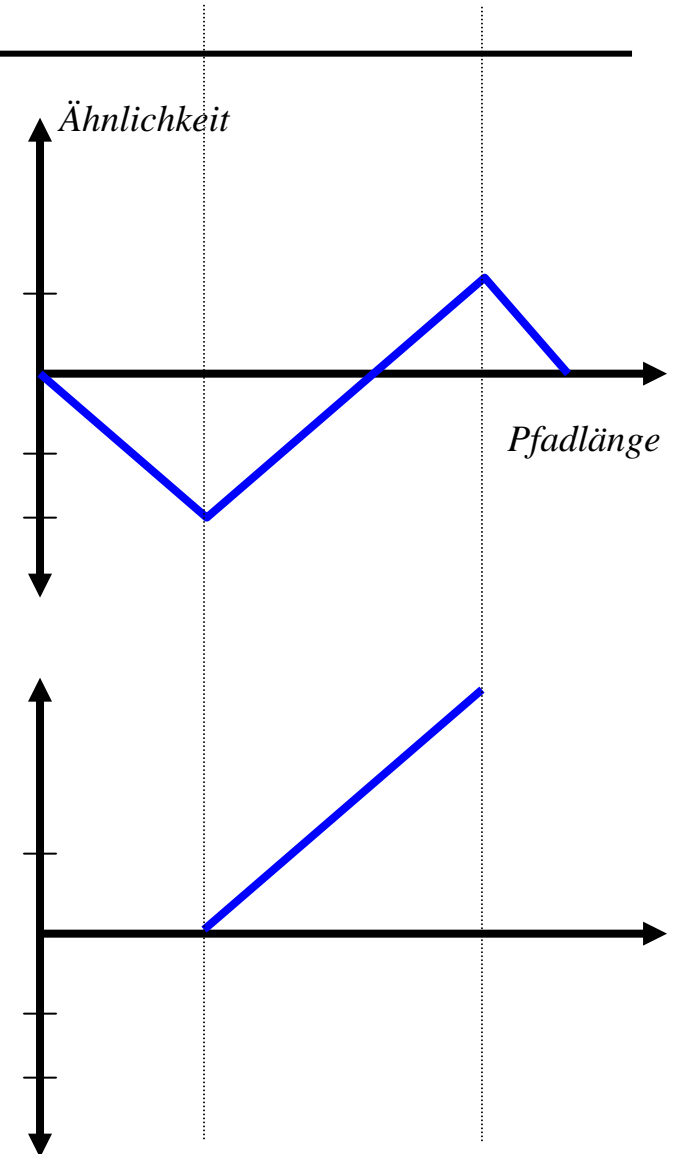
AAAAAAAAAAAAAAAAAA_ AA_ AAAAAA
_____ BBBBBBBBB_ BBBB_ BBBB

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



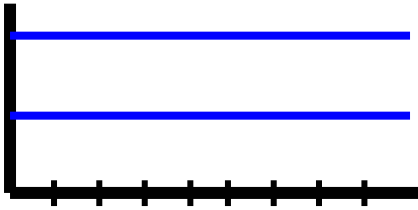
Lösen des lokales Suffixalignmentproblems

- Theorem.
Gegeben Strings A,B. Dann gilt

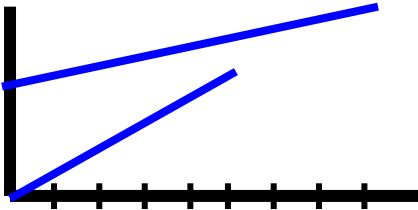
$$v(i, j) = \max \left\{ \begin{array}{l} 0 \\ v(i, j-1) + s(_, B[i]) \\ v(i-1, j) + s(A[i], _) \\ v(i-1, j-1) + s(A[i], B[j]) \end{array} \right\}$$

- Beweisidee
 - 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**

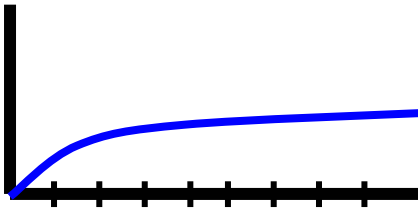
Klassen von Gapscorefunktionen



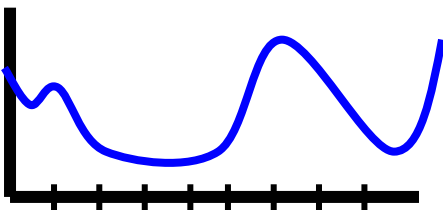
- Konstanter Gapscore



- Linearer Gapscore

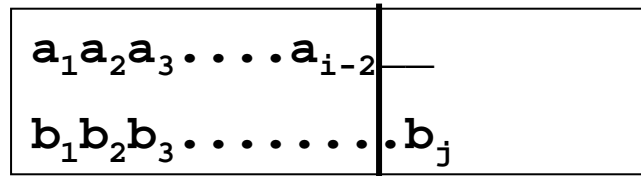
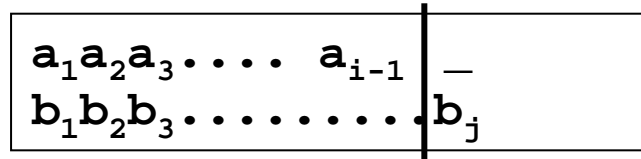


- Konvexer Gapscore

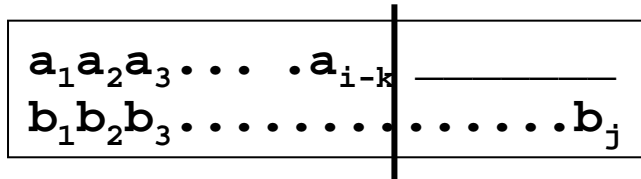


- Beliebiger Gapscore

Mögliche Alignments



Allgemeiner Fall



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

Drei Rekursionsgleichungen



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



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



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

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

Komplexität

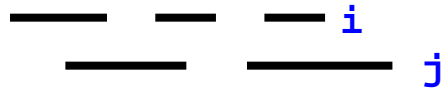
- Theorem

Gegeben Strings A, B mit $|A|=n$, $|B|=m$. Für 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: $E(i,j)$, $F(i,j)$, $G(i,j)$, $V(i,j)$
- G und V sind konstant
- Für E und F müssen die Zellen $(i-1, j-1)$, $(i, 1 \dots j-1)$ und $(1 \dots i-1, j)$ betrachtet werden
- Für fixes i (eine Spalte füllen) betrachten wir also \sum_j 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: $O(n \cdot m^2 + m \cdot n^2)$

Fälle



- Zwei Fälle

- Gap beginnt in A an Position 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

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

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.

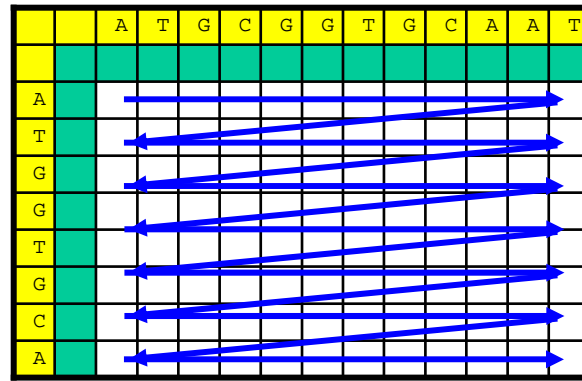
Inhalt dieser Vorlesung

- Alignment in linearem Platz
- K-Banded Alignment
 - Alignment in (meistens) weniger als $O(n \cdot m)$

Alignment mit linearem Platzbedarf

- Bisherige Algorithmen haben $O(m \cdot n)$ Zeit- und $O(m \cdot n)$ Platzbedarf
- Platzbedarf ist ein Problem für Alignierung großer Sequenzen (Genom-Genom)
- Gesucht: **Speicherplatzeffizienterer Algorithmus**

Editabstand in $O(n)$ Space



- Für Zeile $i+1$ sind nur Werte von Zeile i sowie Spalte 0 notwendig
- Aber: Keine Berechnung des tatsächlichen **Alignments** mehr möglich
 - Tracebackinformation ist verloren

Alignment in $O(n)$ Space

- Klassische Erweiterung für Algorithmen basierend auf dynamischer Programmierung
 - Hirschberg: „Algorithms for the longest common subsequence problem“, Journal of the ACM 24, 1977
- Grundidee
 - **Rekursive Zerlegung** des Problems in viele kleinere
 - Diese werden in linearem Platz und quadratischer Zeit gelöst
 - **Gesamtlösung wird aus den Teillösungen** zusammengesetzt
 - Laufzeitkomplexität bleibt gleich
- Beweise: Gusfield, pp. 256-259
 - Wir beschränken uns auf die Idee
 - Wir bezeichnen im Folgenden mit $v(A,B)$ die Ähnlichkeit des optimalen globalen Alignments von A und B

Strings und reverse Strings

- Definition

- Sei A^r das *Reverse des Strings A*
- Für Strings A, B sei $v^r(i, j) = v(A^r[1..i], B^r[1..j])$
 - $A^r[1..i]$ bezeichnet (wie immer) die ersten i Zeichen von A^r , also die letzten i Zeichen von A

- Bemerkung

- Es gilt : $v^r(i, j) = v(A[n-i+1..n], B[m-j+1..m])$
 - Denn: $v(A^r, B^r) = v(A, B)$
- *Berechnung von v^r* kann exakt wie die von v erfolgen

A **ATGCGGT**
B **GGTCGTAG**

A^r **TGGCGTA**
B^r **GATGCTGG**

Problemhalbierung

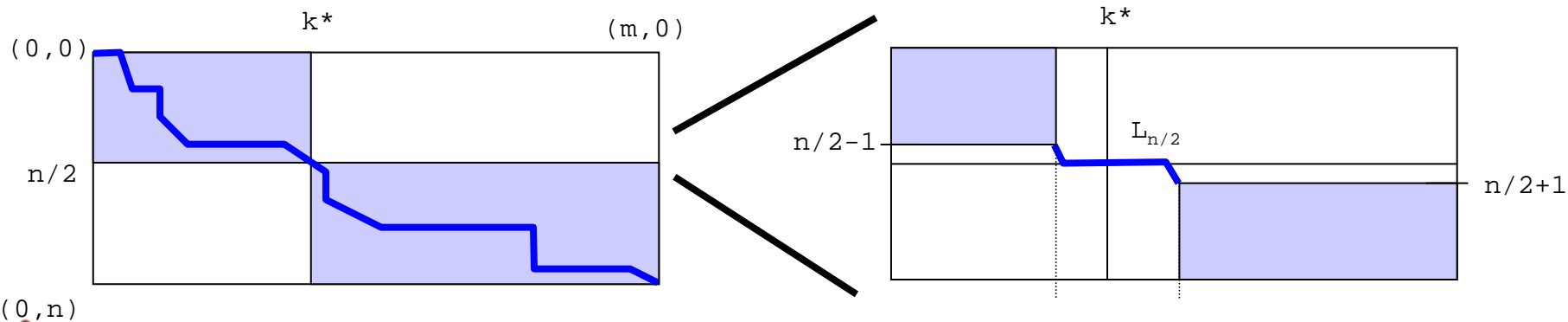
- Lemma
Gegeben zwei Strings A,B. Dann gilt

$$v(n, m) = \max_{0 \leq k \leq m} \left(v(n/2, k) + v^r(n/2, m - k) \right)$$

- Beweisidee
 - Wir alignieren
 - A[1..n/2] mit B[1..k] **vorwärts**
 - A[n/2+1..n] mit B[k+1..m] **rückwärts**
 - Im optimalen Alignment muss irgendein Präfix von B mit A[1..n/2] alignieren
 - Der Rest muss dann mit A[n/2+1..n] alignieren
 - Durch das laufende k erwischen wir auf jeden Fall den richtigen Trennpunkt zwischen Präfix und Suffix
- Bemerkung
 - Das **Problem wird damit bzgl. |A| halbiert**

Teilpfade

- Definition
 - Sei k^* das k , für das das Maximum $v(n,m)$ erreicht wird
 - Sei L der Pfad von $(0,0)$ bis (n,m)
 - Sei $L_{n/2}$ der Pfad zwischen dem letzten Knoten in der Zeile $n/2-1$ und dem ersten Knoten in $n/2+1$
- Lemma
 - L und $L_{n/2}$ müssen k^* enthalten
- Beweisidee: L muss irgendwo die Zeile $n/2$ passieren
 - Die aufwendigen Fallunterscheidungen für gerade / ungerade Zahlen sparen wir uns



Folgerungen

- Lemma
 - k^* kann in *Zeit $O(m \cdot n)$ und Platz $O(m)$* berechnet werden
 - $L_{n/2}$ kann ebenfalls in *Zeit $O(m \cdot n)$ und Platz $O(m)$* berechnet werden
- Beweisidee
 - Berechne Matrix *zeilenweise vorwärts von 0 bis $n/2$* ; speichere jeweils nur die letzte Zeile plus die Tracebackpfeile
 - $O(m \cdot n)$ Zeit, $O(m)$ Platz
 - Berechne Matrix *zeilenweise rückwärts von n bis $n/2$* ; speichere jeweils nur die letzte Zeile plus die Tracebackpfeile
 - $O(m \cdot n)$ Zeit, $O(m)$ Platz
 - Mit den Werten $v(n/2, 1..m)$ und $v^r(n/2, 1..m)$ kann man k^* finden (Summe maximieren)
 - $O(m)$ Zeit, kein Platzverbrauch
 - Der Pfad $L_{n/2}$ wird gefunden durch Traceback von Zelle $(n/2, k^*)$ bis zu einer Zelle in Zeile $n/2-1$ bzw. $n/2+1$
 - $O(m)$ Zeit, kein Platzverbrauch

Beispiel

		A	T	T	G	T	C	G	T	T	T	G	T
A													
T													
G													
G	1	4	5	7	8	6	5	6	5	3	4	3	4
	4	3	2	4	4	5	5	2	1	2	3	5	5
C													
T													
G													

Beispiel

		A	T	T	G	T	C	G	T	T	T	G	T
A													
T													
G													
G	1	4	5	7	8	6	5	6	5	3	4	3	4
	4	3	2	4	4	5	5	2	1	2	3	5	5
C													
T													
G													

K-Band Algorithmus

- Den Platzbedarf haben wir erfolgreich reduziert
- Kann man auch was an der Laufzeit machen?
 - An der Komplexität zum Finden der optimalen Lösung in allen Fällen: Nein
 - Aber oft sucht man nur **besonders gute Alignments**
 - Z.B. Forensik, Vaterschaftstests, Suche nach homologen Gensequenzen
 - Ein Treffer ist entweder fast identisch oder uninteressant

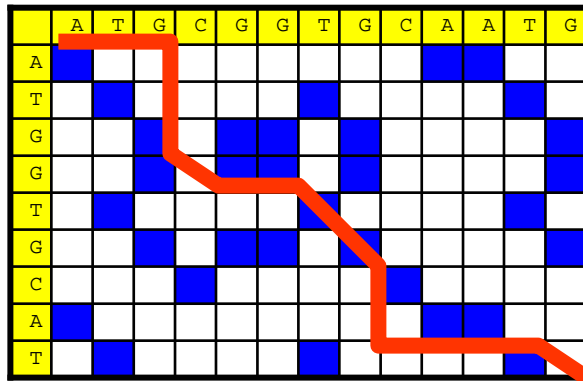
K-Band Algorithmus

- Im Folgenden
 - Wir maximieren globale Ähnlichkeit
 - Sei s ist der Score eines Matches,
 - Sei b der negative Score einer Insertion oder Deletion
 - Kosten für einen Mismatch seien kleiner als b
 - $|A|=|B|=n$
- Folgerung: Für das Alignment von A mit B gilt
 - Der bestmögliche Score ist $n*s$,
 - Der schlechtestmögliche Score ist $2*b*n$

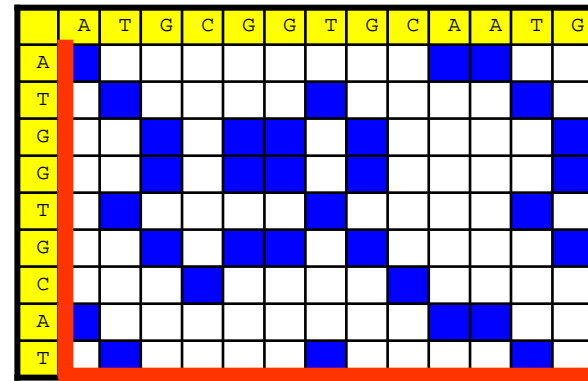
Gute Alignments

- Beobachtung: Gute Alignments **müssen eng an der Hauptdiagonale** bleiben
 - Jedes Abzweigen kostet ein oder mehrere b
 - Und wir müssen auch wieder zurück zur Diagonale – mehr Kosten
- Können wir **nur in der Nähe der Diagonale** bleiben?

ATG___CGGTG__CAATG
 ___ATGG__TGCA____T



_____ATGCGGTGCAATG
 ATGGTGCCAT_____

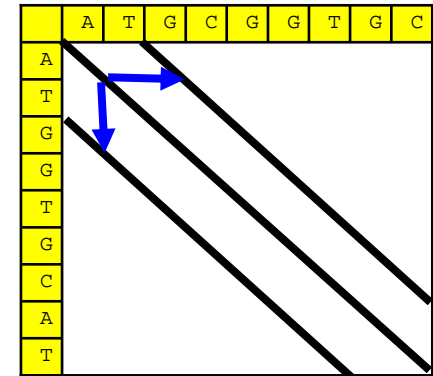


K-Band Algorithmus

- Ansatz: Wir erlauben nur **Abweichungen um maximal $+k/-k$ Schritte**
- Algorithmus
 - Berechnet das beste globale Alignment innerhalb des Bandes der Breite $2*k$

```
for i= 1 to n do
  for j= i-k to i+k do
    if (j<1) or (j>n) continue;
    M[i,j]= M[i-1,j-1] + t(A[i],B[j]);
    if inband(i-1,j) then
      M[i,j]= max( M[i,j], M[i-1,j]+b);
    if inband(i,j-1) then
      M[i,j]= max( M[i,j], M[i,j-1]+b);
  end for;
end for;
return M[n,n]
```

Beispiel: $k=2$



Eigenschaften

- Ein Gap im k -Band kann höchstens $2k$ Leerzeichen lang sein
 - Wenn wir also optimale Alignments mit höchstens z langen Gaps suchen, finden wir die mit $k=z/2$
- Komplexität des K -Band Algorithmus?
 - Für jede Zelle sind maximal 3 Zugriffe, 1 Addition und ein paar Vergleiche notwendig
 - Wir berechnen $O(2k \cdot n)$ Zellen
 - Also: $O(k \cdot n)$
- Können wir vielleicht noch wissen, wie weit weg wir vom Optimum gelandet sind?

Optimalität

- Theorem

Gegeben Strings A, B mit $n=|A|=|B|$. Sei $v_K(A,B)$ der optimale K -Band Score für A und B . Wenn gilt

$$v_K(A,B) \geq s^*(n-k-1) + 2b^*(k+1),$$

dann ist $v_K(A,B) = v(A,B)$.

- Beweis

- Wenn das optimale Alignment im k -Band verläuft, gilt auf alle Fälle $v_K = v(A,B)$
- Wenn nicht, dann muss es irgendwo mindestens einmal aus dem K -Band laufen.
- Im besten solchen Fall haben wir also
 - $n-k-1$ Matches,
 - $k+1$ Insertions (Verlassen des Bandes)
 - $k+1$ Deletions (um am Ende noch (n,n) zu erreichen)

Beweis Fortsetzung

- Theorem

Gegeben Strings A, B mit $|A|=|B|$. Sei $d_K(A,B)$ der optimale K-Band Score für A und B . Wenn

$v_K(A,B) \geq s^(n-k-1) + 2b^*(k+1)$, dann ist $v_K(A,B) = v(A,B)$.*

- Beweis

- ...

- Im besten solchen Fall haben wir $n-k-1$ Matches und $k+1$ Insertions und weitere $k+1$ Deletions

- Der **bestmögliche Score für ein optimales Alignment, dass das K-Band mindestens einmal verlässt**, ist also $s^*(n-k-1) + 2b^*(k+1)$

- Wenn also $v_K \geq s^*(n-k-1) + 2b^*(k+1)$, muss das optimale Alignment im K-Band laufen

- Dann ist es auch durch den K-Band Algorithmus gefunden worden

- qed.

Iteratives K-Band

- Das können wir ausnutzen, um das optimale Alignment **iterativ** zu finden
- Wir verdoppeln k in jedem Schritt, solange, bis wir garantiert das optimale Alignment haben

```
k = 1;
while (true) do
    compute  $v_k$ ;           // Costs  $O(k*n)$ 
    if  $v_k \geq s(n-k-1)+2b(k+1)$  then
        return  $v_k$ ;
    else
        k = 2*k;
    end if;
end while;
```

Komplexität

- Theorem.
Sei $v=v(A,B)$. Der iterative K-Band Algorithmus benötigt $O(sn^2-vn)$ Laufzeit.
- Beweis
 - Beachte: v_k wird mit wachsendem k nie kleiner
 - Der Algorithmus stoppt, wenn $v_k \geq s(n-k-1)+2b(k+1)$, also
$$k \geq \frac{sn - v_k}{s - 2b} - 1$$
 - Bis dahin wurden $O(1n+2n+4n+\dots+kn) \sim O(2kn)$ Berechnungen durchgeführt
 - Wenn wir bei k stoppen, dann kann bei $k/2$ die Abbruchbedingung noch nicht erfüllt gewesen sein, und damit gilt:

$$\frac{k}{2} < \frac{sn - v_{k/2}}{s - 2b} - 1$$

Komplexität - 2

– Betrachten wir den vorletzten Schritt $k/2$. Zwei Möglichkeiten

- $V_{k/2} = V_k = v$ $k < 2 \left(\frac{sn - v}{s - 2b} - 1 \right)$

- $V_{k/2} < V_k = v$. Dann haben wir mit $k/2$ das optimale Alignment noch nicht gefunden, weil es mehr als $k/2$ Indels hat. Damit muss gelten:

$$v \leq s(n - k/2 - 1) + 2b(k/2 + 1)$$

- Und damit: $k \leq 2 \left(\frac{sn - v}{s - 2b} - 1 \right)$

– Die Berechnungszeit $2kn$ können wir damit nach oben abschätzen durch:

$$2nk \leq 2n * 2 \left(\frac{sn - v}{s - 2b} - 1 \right) = 4n \left(\frac{sn - v}{s - 2b} - 1 \right)$$

– Und das ist $O(sn^2 - vn)$

– qed.



Komplexität K-Band?

- Aber ...
 - $O(sn^2 - vn)$ konvergiert gegen 0 wenn v gegen $s \cdot n$ konvergiert (also die Sequenzen identisch sind)
 - Das sollte es nicht – denn auch bei identischen Sequenzen benötigen wir $O(n)$, um den Score zu berechnen
 - Wir wissen ja vorher nicht, dass die Sequenzen identisch sind!
 - Also?
- Aber ...
 - Abschätzung basiert auf dem letzten ausgeführten Schritt $k/2$ und schätzt damit k ab
 - Wenn aber Sequenzen identisch sind, gibt es nur einen Schritt
 - Konvergenz „ v gegen n “ darf nicht angenommen werden; k muss ganze Zahl bleiben

Zusammen

- K-Band Algorithmus hat Komplexität $O(sn^2 - vn)$
- Setzen wir z.B. $s=1$
 - Dann kann v maximal n sein
 - Sehr ähnliche Sequenzen erreichen Wert nahe bei v
 - Dann läuft der Algorithmus auch sehr schnell
- K-Band also umso besser, je **ähnlicher die Sequenzen** sind
 - Gut, um das schnell festzustellen (Algorithmus mit kleinen k laufen lassen)
 - Schlecht, um „irgendwelche“ Alignments zu berechnen