Algorithmische Bioinformatik

Alignment in linearem Platz

K-Band: Schnelles globales Alignment

Ulf Leser

Wissensmanagement in der Bioinformatik

Ziele

- Zwei Tricks zur schnelleren / kompakteren Berechnung sehr großer Alignments
- Einstieg: Schnellere Verfahren für häufige Situationen
  - Kein klassischer Average Case

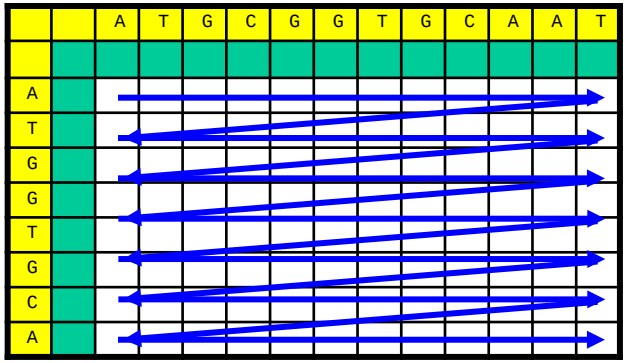
Inhalt dieser Vorlesung

- Alignment in linearem Platz
- K-Band Alignment

Alignment mit linearem Platzbedarf

- Alle bisherigen Alignment-Algorithmen
  - O(m*n) Zeitkomplexität
  - O(m*n) Platzkomplexität
- Gerade der Platzbedarf ist ein großes Problem für Alignierung großer Sequenzen (Genom-Genom)
- Gesucht: Speicherplatzeffizienter Algorithmus
  - ... für globales Alignment

Editabstand in O(n) Space



- Für Zeile i+1 sind nur Werte von Zeile i sowie Spalte 0 notwendig
  - Also Platzbedarf O(min(m,n))
- Aber: Keine Berechnung des Alignments mehr möglich
  - Tracebackinformation ist verloren

Alignment in O(n) Space

- Klassische Erweiterung
  - Hirschberg: „Algorithms for the longest common subsequence problem“, Journal of the ACM 24, 1977
  - Funktioniert für viele Algorithmen mit dynamischer Prg.
- Berechnung von einem optimalen Alignment (nicht allen)
- Grundidee
  - Rekursive Zerlegung des Problems
  - Kleinere Probleme lösen in linearem Platz und quadratischer Zeit
  - Gesamtlösung wird aus Teillösungen zusammengesetzt
  - Platzkomplexität sinkt, Laufzeitkomplexität gleich
- Vereinfachungen
  - Sei $2^z=n=|A|$; vermeidet lange Formeln
  - Wir nehmen an, dass das optimale Alignment eindeutig ist

Strings und reverse Strings

- Definition
  - *Sei A^r das Reverse des Strings A*
  - *Für Strings A,B sei* $d^r(i,j)=d(A^r[1..i], B^r[1..j])$
- Bemerkungen
  - $A^r[1..i]$ sind die letzten i Zeichen von A
  - $d(A^r, B^r) = d(A, B)$
  - $d^r(i,j) = d(A[n-i+1..n], B[m-j+1..m])$
  - Berechnung von d^r kann exakt wie die von d erfolgen

```
A  .............ATGCGGT
B  ................GGTCGTAG

Ar  TGGCGTA............
Br  GATGCTGG................
```

Problemhalbierung

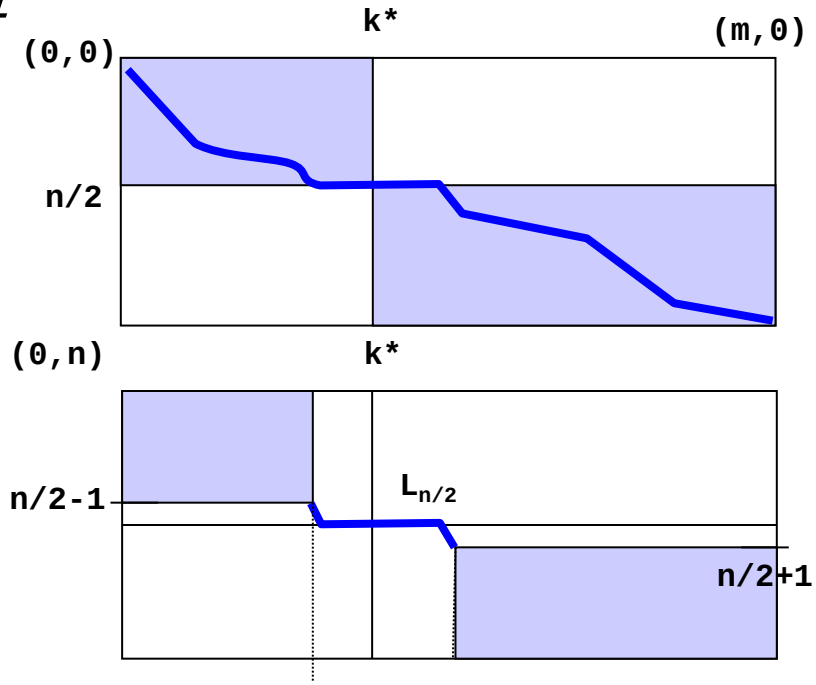
- Lemma
  Gegeben A, B. Dann gilt

$$d(n,m) = \min_{0 \le k \le m} \left(d(n/2,k) + d^r(n/2, m-k) \right)$$

- Beweis
  - Wir alignieren
    - A[1..n/2] mit B[1..k] vorwärts
    - A[n/2..n] mit B[k+1..m] rückwärts
  - Im optimalen Alignment aligniert ein Präfix von B mit A[1..n/2]
  - Der Rest muss mit A[n/2+1..n] alignieren
  - k läuft über alle möglichen Präfixlängen
- Beobachtung: Das Problem wird bzgl. |A| halbiert

Teilpfade

- Definition
  - *Sei k* ein k, das das Minimum bei Berechnung von d(n,m) erzeugt*
  - *Sei L der optimale Pfad von (0,0) bis (n,m)*
  - *Sei $L_{n/2}$ der optimale Pfad zwischen der letzten Zelle in Zeile n/2-1 und der ersten Zelle in Zeile n/2+1*
- Lemma
  L und $L_{n/2}$ müssen k enthalten*
- Beweisidee
  - L muss bei k* Zeile n/2 passieren
  - L enthält $L_{n/2}$
- [Gibt es mehrere optimale Pfade, muss das Lemma für einen davon gelten]



Folgerungen

- Lemma
  - *k* kann in Zeit O(m*n) und Platz O(m) berechnet werden*
  - *$L_{n/2}$ kann in Zeit O(m*n) und Platz O(m) berechnet werden*
- Beweis per Konstruktion
  - Berechne Matrix zeilenweise vorwärts von 0 bis n/2; speichere jeweils nur die letzte Zeile plus die Tracebackpfeile nur dieser Zeile
    - O(m*n) Zeit, O(m) Platz
  - Berechne Matrix zeilenweise rückwärts von n bis n/2; speichere jeweils nur die letzte Zeile plus die Tracebackpfeile dieser Zeile
    - O(m*n) Zeit, O(m) Platz
  - Finde k* als Minimum von $d(n/2, k)+d^r(n/2, m-k)$
    - O(m) Zeit, O(1) Platz
  - Finde Pfad $L_{n/2}$ durch Traceback von Zelle (n/2,k*) bis zu einer Zelle in Zeile n/2-1 und bis zu einer Zelle in Zeile n/2+1
    - O(m) Zeit, O(n+m) Platz (über alle rekursiven Aufrufe – maximale Pfadlänge)

Beispiel

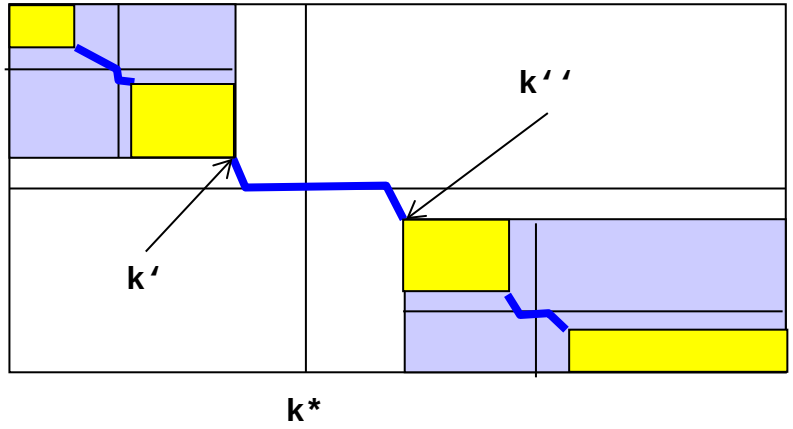
		A	T	T	G	T	C	G	T	T	T	G	T
A													
T													
G													
G	2 9	4 7	5 6	6 5	6 4	7 5	7 3	9 2	9 2	... ...	... ...	... ...	... ...
C													
T													
G													

Beispiel

		A	T	T	G	T	C	G	T	T	T	G	T
A													
T													
G													
G	29	47	56	65	64	75	73	92	92	... ...	... ...	... ...	... ...
C													
T													
G													

Rekursion

- Wo sind wir?
  - Wir halten die gewünschten Komplexitätsschranken
  - Wir haben das Mittelstück von L berechnet
- Damit können wir rekursiv absteigen
  - Löse rekursiv die Probleme
    - für A[1..n/2-1] / B[1..k‘-1] und
    - für A[n/2+1..n] / B[k“+1..m]
  - k‘ / k“: Punkte, wo $L_{n/2}$ die Zeile n/2-1 / n/2+1 trifft
- Platzkomplexität steigt nicht
- Zeitkomplexität ...



Zeitkomplexität (sei m=n)

- A wird in jedem Schritt halbiert
- Im schlimmsten Fall wird B immer nur um eins kleiner
- Damit ergibt sich die Laufzeitkomplexität als

$$n*n+\left(\frac{n}{2}*1+\frac{n}{2}(n-1)\right)+\left(\frac{n}{4}*2+\frac{n}{4}(n-2)\right)+\ldots+\left(\frac{n}{2}+\frac{n}{n}(n-\log(n))\right)$$

$$=n^2+\log(n)*n/2+\left(\frac{n^2}{2}+\frac{n^2}{4}+\ldots\frac{n^2}{n}\right)-\left(\frac{n}{2}+\frac{2n}{4}+\frac{3n}{8}\ldots+\frac{\log(n)*n}{n}\right)$$

$$\leq n^2+n^2/2+n^2/2+n^2/4\ldots$$

$$=n^2\sum_{i=1..\log(n)}2^{-i}=O(n^2)$$

Inhalt dieser Vorlesung

- Alignment in linearem Platz
- k-Band Alignment
  - Globales Alignment in (meistens) weniger als O(n*m)

Beobachtung

- Kann man auch die Zeitkomplexität reduzieren?
  - Zum Finden der optimalen Lösung in allen Fällen: Nein
    - (Beim lokalen Alignment: Auch nein)
  - Für typische Anwendungen (Sequenzen sehr ähnlich): Ja
    - Oft sucht man nur besonders gute Alignments
      - z.B. Suche nach homologen Gensequenzen
    - Vergleich ergibt also entweder „fast identisch“ oder „uninteressant“
  - Wenn wir schnell feststellen können, dass zwei Sequenzen nicht sehr ähnlich sind, können wir oft die exakte Berechnung der Ähnlichkeit sparen

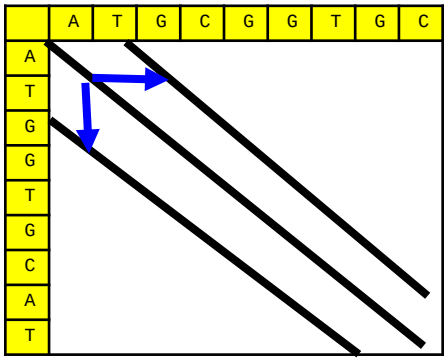
Voraussetzungen

- Im Folgenden
  - Wir maximieren globale Ähnlichkeit
  - Sei s der uniforme, positive Score eines Matches
    - Individuelle Scores – viel längere Formeln
  - Sei b der uniforme, negative Score einer Insertion / Deletion
  - Kosten für Mismatch seien kleiner als 2*b
  - Sei |A|=|B|=n
- Folgerung: Für das Alignment von A mit B gilt
  - Der bestmögliche Score ist n*s
  - Der schlechteste mögliche Score ist 2*b*n

Gute Alignments

- Ähnliche Sequenzen haben gute Alignments
- Gute Alignments sind eng an der Hauptdiagonale
  - Jedes Abzweigen kostet mindestens 2*b: Weg von der Diagonale und wieder hin
- k-Band Algorithmus Idee
  - Berechne zunächst nur Alignments, die eng (maximaler Abstand k) an der Diagonale bleiben
  - Wenn diese tatsächlich optimal sind – fertig
    - Das müssen wir irgendwie prüfen
  - Wenn nicht – k vergrößern und iterieren

Beispiel: k=2

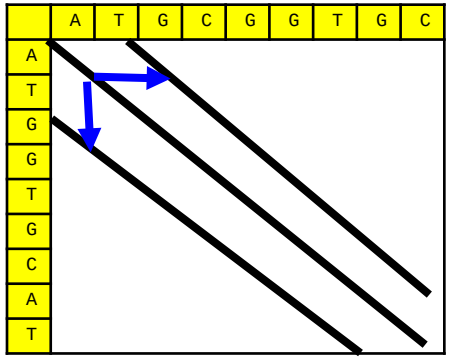


Algorithmus

- Abweichungen um maximal +k/-k Schritte
- Algorithmus
  - Berechnet das beste globale Alignment innerhalb eines Bandes um die Hauptdiagonale 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



Komplexität

- Komplexität des K-Band Algorithmus?
  - Für jede Zelle sind maximal 3 Zugriffe, 1 Addition und ein paar Vergleiche notwendig
  - Wir berechnen O(2k*n) Zellen
  - Also: O(k*n)
- Können wir abschätzen, ob wir schon optimal sind?

Optimalität

- Theorem
  Geg. A, B mit |A|=|B|=n. Sei $v_k(A,B)$ der optimale k-Band Score für A und B. Dann gilt:
  $$v_k(A,B)=v(A,B) \text{ wenn } v_k(A,B)\geq s*(n-k-1)+2b*(k+1)$$
- Beweis
  - Wenn das optimale Alignment im k-Band verläuft, gilt $v_K=v(A,B)$
  - Wenn nicht, muss es mindestens einmal aus dem k-Band laufen
  - Im besten solchen Fall haben wir
    - n-k-1 Matches
    - k+1 Insertions (Verlassen des Bandes)
    - k+1 Deletions (zurück zur Diagonale)

Beweis Fortsetzung

- Theorem
  Geg. A, B mit |A|=|B|=n. Sei $v_k(A,B)$ der optimale k-Band Score für A und B. Dann gilt:
  $$v_k(A,B)=v(A,B) \text{ wenn } v_k(A,B)\geq s*(n-k-1)+2b*(k+1)$$
- Beweis
  - Im besten Fall außerhalb des k-Bandes haben wir n-k-1 Matches und 2*(k+1) Insertions / Deletions
  - Der bestmögliche Score für ein optimales Alignment, dass das k-Band mindestens einmal verlässt, ist daher s*(n-k-1)+2b*(k+1)
  - Wenn also $v_k(A,B) \geq s*(n-k-1)+2b*(k+1)$, muss das optimale Alignment im k-Band laufen
  - qed.

Iteratives K-Band

- Iterative Bestimmung des optimalen Alignments
  - Verdopple k in jedem Schritt; tue das solange, bis garantiert das optimale Alignment gefunden wurde

```
k = 1;
while (true) do
        compute v_k;                // Costs O(k*n)
        if v_k ≥ 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 hat eine Zeitkomplexität von O(sn²-vn).
- Beweis
  - Beachte: v_k wird mit wachsendem k nie kleiner
  - Algorithmus stoppt bei dem k‘,
    bei dem $v_{k'} \geq s(n-k'-1)+2b(k'+1)$, also $k' \geq \frac{sn - v_{k'}}{s-2b} - 1$
  - Bis dahin hatten wir O(1n+2n+4n ...+k‘n) = O(2k‘n) Berechnungen
  - Wenn wir bei k‘ stoppen, kann bei k‘/2 die Abbruchbedingung noch nicht erfüllt gewesen sein. Damit gilt:

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

Komplexität – 2

- Betrachten wir den vorletzten Schritt k‘/2. Zwei Möglichkeiten
  - $v_{k'/2}=v_{k'}=v$. Wir haben schon das optimale Alignment, merken es aber nicht. Also $$k' < 2\left(\frac{sn-v}{s-2b}-1\right)$$
  - $v_{k'/2}<v_{k'}=v$. Dann haben wir bei k‘/2 das optimale Alignment noch nicht gefunden, weil es mehr als k‘/2 Insdels hat. Damit muss gelten
    $$v \le s(n-k'/2-1)+2b(k'/2+1)$$
  - Zusammen: $$k' \le 2\left(\frac{sn-v}{s-2b}-1\right)$$
- Die Berechnungszeit 2k‘n können wir nun nach oben abschätzen
  $$2nk' \le 2n*2\left(\frac{sn-v}{s-2b}-1\right)=4n\left(\frac{sn-v}{s-2b}-1\right)$$
- Das ist $O(sn^2 - vn)$
- qed.

Komplexität K-Band?

- Aber …
  - $O(sn^2-vn)$ konvergiert gegen 0 wenn v gegen s*n konvergiert
    - Also die Sequenzen fast 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“ ist nicht legitim, weil k ganze Zahl sein muss

Folgerungen

- Algorithmus ist in $O(sn^2 - vn)$
- 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 ist umso schneller, je ähnlicher die Sequenzen sind
  - Gut, um schnell auf hohe Ähnlichkeit zu testen
    - Über k iterieren und abbrechen, wenn Scores nicht mehr gut genug werden können
  - Schlecht, um „irgendwelche“ Alignments zu berechnen

Selbsttest

- Wie kann man ein optimales Alignment in linearem Platz berechnen?
- Wie viel Platz braucht man denn dabei genau?
- Wann lohnt es sich (wenn überhaupt), statt an der n- an der m-Kante zu halbieren?
- Warum verdoppelt der k-Band Algorithmus immer k? Gäbe es andere Strategien?
- Wie schnell kann der k-Band Algorithmus maximal laufen? Könnte es theoretisch noch schneller gehen?