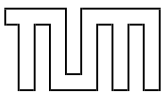
# Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2009



# Übersicht

# Konkave Lückenstrafen

Vereinfachte Notation für eine feste Zeile $i$ der Tabelle der dynamischen Programmierung:

$$\begin{aligned} D(j) &:= D[i,j], \\ E(j) &:= \min\{D(k) + g(j-k) \ : \ k \in [0:j-1]\}, \\ \mathrm{Cand}(k,j) &:= D(k) + g(j-k). \end{aligned}$$

Dann gilt:

$$E(j) = \min\{\mathrm{Cand}(k,j) \ : \ k \in [0:j-1]\}.$$

# Konkave Lückenstrafen

- Berechnung von $E$ mit dem folgenden Algorithmus
- Bei Bestimmung von $E(j)$ wird darauf verzichtet, rückwärts alle Alignments mit einem Einfüge-Block zu betrachten.
- Stattdessen wird jedes Mal, wenn der Wert von $D(j)$ berechnet wird, der bislang optimale Wert von $E(j')$ berechnet (in $\hat{E}(j')$), der durch eine Einfüge-Lücke von $j+1$ bis $j'$ entsteht.
- Nicht nur der bislang beste Score $\hat{E}(j')$ wird gespeichert, sondern auch die linkeste Position $b(j')$, bei der für einen optimalen Score die Lücke beginnt.

# Konkave Lückenstrafen

**Algorithmus 18** : Forward_Propagation()

**for** $(j := 1; j \leq m; j{+}{+})$ **do**
    $\hat{E}(j) := \mathrm{Cand}(0, j)$;
    $b(j) := 0$;

**for** $(j := 1; j \leq m; j{+}{+})$ **do**
    $E(j) := \hat{E}(j)$;
    $D(j) := \min\{E(j), F(j), G(j)\}$;
    **for** $(j' := j + 1; j' \leq m; j'{+}{+})$ **do**
        **if** $(\hat{E}(j') > Cand(j, j'))$ **then**
            $\hat{E}(j') := \mathrm{Cand}(j, j')$;
            $b(j') = j$;

# Konkave Lückenstrafen

Dieses Vorgehen bringt für eine Zeile noch keine Zeiteinsparung.

Das folgende Lemma zeigt jedoch, dass für ein $j$ nicht unbedingt alle $j' > j$ betrachtet werden müssen:

## Lemma

*Sei* $k < j < j' < j'' \in [1 : m]$.

*Wenn* $Cand(k, j') \leq Cand(j, j')$ *gilt, dann ist* $Cand(k, j'') \leq Cand(j, j'')$.

# Konkave Lückenstrafen

## Beweis.

Sei

$$\begin{aligned} q' &= j' - j \geq 0 \\ q'' &= j'' - j \geq q' \\ d &= j - k > 0 \end{aligned}$$

Weil $g$ konkav ist, gilt:

$$g(q' + d) - g(q') \geq g(q'' + d) - g(q'')$$

bzw.

$$g(j' - k) - g(j' - j) \geq g(j'' - k) - g(j'' - j)$$

# Konkave Lückenstrafen

**Beweis.**

Nach Multiplikation mit $-1$:

$$g(j' - j) - g(j' - k) \leq g(j'' - j) - g(j'' - k) \tag{1}$$

Weiterhin gilt wegen $\mathrm{Cand}(k, j') \leq \mathrm{Cand}(j, j')$:

$$D(k) + g(j' - k) \leq D(j) + g(j' - j) \tag{2}$$

# Konkave Lückenstrafen

**Beweis.**

Somit gilt:

$$\begin{aligned}
\mathrm{Cand}(k, j'') &= D(k) + g(j'' - k) \\
&\quad \text{wegen Ungleichung 2} \\
&\leq D(j) + g(j' - j) - g(j' - k) + g(j'' - k) \\
&\quad \text{wegen Ungleichung 1} \\
&\leq D(j) + g(j'' - j) \\
&= \mathrm{Cand}(j, j'').
\end{aligned}$$

Damit ist das Lemma bewiesen. □

# Konkave Lückenstrafen

Aus diesem Lemma folgt sofort das folgende Korollar, das die Abbruch-Bedingung in der for-Schleife im letzten Algorithmus einschränkt.

## Folgerung

*Wenn* $Cand(j, j') \geq \hat{E}(j')$ *für ein* $j' > j$ *gilt, dann gilt für alle* $j'' > j'$ *die Beziehung* $Cand(j, j'') \geq \hat{E}(j'')$.

# Konkave Lückenstrafen

- jedoch: noch keine wesentliche Zeitersparnis
- Das folgende Lemma gibt jedoch weiteren Aufschluss, an welchen Positionen in einer Zeile die besten Einfüge-Lücken beginnen.

**Lemma**

*Sei* $j \in [0 : m]$. *Zum Zeitpunkt, wenn* $D(j)$ *aktualisiert wurde, aber* $\hat{E}(j')$ *noch nicht, gilt*

$$b(j') \geq b(j'+1) \quad \textit{für alle } j' \in [j+1 : m]$$

# Konkave Lückenstrafen

## Beweis.

Es gilt nach Definition von $b(j')$:

$$\mathrm{Cand}(b(j'), j') \leq \mathrm{Cand}(b(j'+1), j').$$

Annahme für Widerspruchsbeweis: $b(j') < b(j'+1)$
Vor.: $k = b(j') < j = b(j'+1) < j' < j'' = j'+1$
Nach letztem Lemma:

$$\mathrm{Cand}(b(j'), j'+1) \leq \mathrm{Cand}(b(j'+1), j'+1).$$

Dann wäre jedoch $b(j'+1) \leq b(j')$, da immer der linkest mögliche Index für den bislang optimalen Score in $b(\cdot)$ gesetzt wird.
Also ist $b$ monoton fallend. □

# Konkave Lückenstrafen

**Definition**

Die Partition von $[j+1 : m]$ in disjunkte Intervalle mit gleichen $b(\cdot)$-Wert wird Block-Partition genannt.

- Betrachten wir zunächst den Fall $j = 1$ und die zugehörige triviale Blockpartition $[2 : m]$ von $[2 : m]$, da $b(j) = 0$ für alle $j \in [2 : m]$ ist.
- Nach dem Setzen von $E(1)$ mittels $\hat{E}(1)$ kann sich die Block-Partition ändern.
- Dabei können sich nur einige Werte von 0 auf 1 ändern.
- Nach dem vorherigen Lemma müssen diese einen Block aus den ersten Elementen der monoton fallenden Folge $b$ bilden.

# Konkave Lückenstrafen

**Fall 1** $b(2) = \cdots b(m) = 0$ unverändert. Nach der letzten Folgerung ist dies genau dann der Fall, wenn $\mathrm{Cand}(1,2) \geq \hat{E}(2) = \mathrm{Cand}(0,2)$ ist.

**Fall 2** $b(2) = \cdots b(k) = 1$ und $b(k+1) = \cdots b(m) = 0$. Dies ist genau dann der Fall, wenn $\mathrm{Cand}(1,j') < \hat{E}(j')$ für alle $j' \in [2:k]$ und $\mathrm{Cand}(1,j') \geq \hat{E}(j')$ für alle $j' \in [k+1:m]$.

**Fall 3** $b(2) = \cdots b(m) = 1$. Dies kann nur genau dann der Fall sein, wenn $\mathrm{Cand}(1,j') < \hat{E}(j')$ für alle $j' \in [2:m]$.

Die Ermittlung von $k$ im Fall 2) lässt sich mit Hilfe einer binären Suche auf dem Intervall $[2:m]$ mit der Funktion $\mathrm{Cand}(1,j') - \hat{E}(j')$ bewerkstelligen.
Ist $\mathrm{Cand}(1,j') - \hat{E}(j') \geq 0$, sucht man in der linken Hälfte weiter, andernfalls in der rechten.

# Konkave Lückenstrafen

$j \geq 2$:

- Die Blockstruktur von $(b_{j+1}, \ldots, b_m)$ sei durch eine Liste von Indizes $(p_1, \ldots, p_r)$ gegeben, d.h.

$$b(j+1) = \cdots = b(p_1) > b(p_1+1) \cdots$$
$$\cdots b(p_{r-1}) > b(p_{r-1}+1) = \cdots = b(p_r) = b(m)$$

- Sei $b_i = b(p_i)$
- Nachdem $E(j)$ und $D(j)$ aktualisiert wurden, erfolgt die Forward-Propagation.

# Konkave Lückenstrafen

**Fall 1** Sei

$$\mathrm{Cand}(k, j+1) = \hat{E}(j+1) \leq \mathrm{Cand}(j, j+1)$$

für ein $k$, das $\hat{E}(j+1)$ bisher definiert, also $k = b(j+1) = b_x$ für ein geeignetes $x \in [1 : r]$.

Nach dem Lemma gilt dies genau dann, wenn $\mathrm{Cand}(k, j') = \hat{E}(j') \leq \mathrm{Cand}(j, j')$ für alle $j' > j$.

Die Blockpartition bleibt in diesem Fall unverändert.

# Konkave Lückenstrafen

**Fall 2** Sei nun

$$\hat{E}(j+1) > \mathrm{Cand}(j, j+1)$$

Vergleiche nacheinander $\hat{E}(p_i)$ mit $\mathrm{Cand}(j, p_i)$ bis die Block-Partition abgearbeitet wurde oder $\hat{E}(p_s) \leq \mathrm{Cand}(j, p_s)$ gilt.
Im 1. Fall wird $[j+1 : m]$ ein Block mit $p_1 = m$ und $b_1 = j$.
Im 2. Fall werden die Blöcke $[1 : p_1]$ mit $[p_{s-2}+1 : p_{s-1}]$ zu einem Block verschmolzen.
Eventuell gehört noch ein Anfangsstück des Intervalls $[p_{s-1}+1 : p_s]$ zu diesem Block, der Rest des Blocks überlebt als eigener Block der neuen Block-Partition.
Dieses Anfangsstück ermitteln wir wie im Falle für $j = 1$ mit einer binären Suche (in diesem Block sind $b$-Werte wieder alle gleich).

# Konkave Lückenstrafen

- Wir halten fest, dass $\hat{E}(j) = \mathrm{Cand}(b(j), j)$ gilt.
- Weiterhin wissen wir, dass im Block $[p_{k-1} + 1 : p_k]$ $b(j) = b(p_k)$ für alle $j \in [p_{k-1} + 1 : p_k]$ gilt.
- Also gilt $\hat{E}(j) = \mathrm{Cand}(b_k, j)$, wenn sich $j$ im $k$-ten Block befindet.
- Wir brauchen daher $\hat{E}$ nicht explizit speichern, sondern können die Werte mit Hilfe der Liste der $b$-Werte zur Liste der Blockgrenzen $p_i$ der aktuellen Block-Partition in konstanter Zeit berechnen.
- Auch die $b$-Werte speichern wir nicht für jedes $i$, sondern nur einmal für jeden Block der Block-Partition als Liste.

# Konkave Lückenstrafen

- Laufzeit der binären Suchen in jeder Phase für $j \in [1 : m]$: $\mathcal{O}(\log(m))$
- Alle binären Suchen einer Zeile der Tabelle der dynamischen Programmierung: Zeit $\mathcal{O}(m \log(m))$
- Wieviel verschiedene Blöcke der Blockpartition werden innerhalb einer Zeile geprüft?
- Werden in einer Block-Partition maximal 2 Blöcke inspiziert, so werden maximal 2 Test ausgeführt und die Blockpartition verlängert sich anschließend um höchstens 1.
- Werden mindestens $\ell > 2$ Blöcke einer Block-Partition inspiziert, so werden anschließend aus diesen $\ell$ Blöcken maximal 2.
- Damit verringert sich die Anzahl der Blöcke einer Block-Partition um mindestens $\ell - 2$.

# Konkave Lückenstrafen

- Für jeden Durchgang $j \in [1 : m]$ rechnen wir zuerst zwei Block-Inspektionen direkt ab, das ergibt Aufwand von $\mathcal{O}(m)$.
- Wenn es mehr als zwei Inspektionen gab, so sind diese proportional zur Anzahl der eliminierten Blocks.
- Da wir mit einem Block beginnen und für jedes $j \in [1 : m]$ maximal einen neuen Block generieren können und auch nur so viele Blöcke eliminiert werden können, wie erzeugt werden, können maximal $\mathcal{O}(m)$ Blöcke eliminiert werden.
- Somit können maximal $\mathcal{O}(m)$ Tests auf Blöcken ausgeführt werden.

# Konkave Lückenstrafen

- Damit sind für die Berechnung der $E$-Werte maximal $O(mn\log(m))$ Operationen nötig.
- Dieselbe Argumentation gilt auch für die $F$-Werte.
- Hierbei ist nur zu beachten, dass die $F$-Werte zwar pro Spalte gelten, aber ebenfalls zeilenweise berechnet werden.
- Damit sind für jede Spalte die Listen $(p_i)_{i\in[1:r]}$ der aktuellen Block-Partition und die Liste $(b_i)_{i\in[1:r]}$ der zugehörigen $b$-Werte zu speichern.
- Dennoch werden pro Spalte maximal $n\log(n)$ Operationen ausgeführt, also insgesamt $\mathcal{O}(mn\log(n))$.

# Konkave Lückenstrafen

**Theorem**

*Seien $s, t \in \Sigma^*$ mit $n = |s|$ und $m = |t|$.*

*Ein optimales globales paarweises Sequenzen-Alignment für $s$ und $t$ mit konkaven Lückenstrafen lässt sich in Zeit $\mathcal{O}(nm \log(n+m))$ berechnen.*

**Beweis.**

Wir haben bereits gezeigt, dass die Laufzeit $\mathcal{O}(mn(\log(m) + \log(n)))$ beträgt.

Mit $\mathcal{O}(mn(\log(m) + \log(n))) \subseteq \mathcal{O}(mn \log(n+m))$ folgt die Behauptung. □