technische universität
dortmund

Algorithmen auf Sequenzen

Paarweises Sequenzalignment

Dominik Kopczynski

Lehrstuhl für Algorithm Engineering (LS11)
Fakultät für Informatik
TU Dortmund

Paarweises Sequenzalignment

- Bisher wurde das globale Alignment zur Berechnung der Edit-Distanz und das semiglobale Alignment für fehlertolerantes Pattern-Matching vorgestellt.
- Jedoch konnte bisher nicht der String-Homomorphismus wiedergegeben werden.
- Das “eigentliche” Alignment steht also noch aus.

Globales Alignment

- Mit Hilfe des Traceback-Verfahrens kann die optimale Reihenfolge an Edit-Operationen ermittelt werden.
- Für jede Zelle aus der bekannten DP-Matrix muss nachgehalten werden, aus welcher Vorgängerzelle der bisherige minimale Wert kommt.
- Da es nur drei Möglichkeiten gibt, erhöht sich die Gesamtlaufzeit nicht, bleibt also bei $\mathcal{O}(mn)$.
- Der Speicherverbrauch steigt jedoch auf $\mathcal{O}(mn)$, da nun die gesamte Matrix nachgehalten werden muss.

Traceback-Verfahren

- Der String-Homomorphismus A beginnt mit einem leeren Array. Ausgehend vom Feld $D[m][n]$ wird dessen Vorgänger ermittelt.
- Bei einem diagonalen Sprung $(i, j) \rightarrow (i-1, j-1)$, findet ein Match/Mismatch statt. Es wird der Eintrag $(s[i-1], t[j-1])$ zu A hinzugefügt.

Traceback-Verfahren

- Der String-Homomorphismus A beginnt mit einem leeren Array. Ausgehend vom Feld $D[m][n]$ wird dessen Vorgänger ermittelt.
- Bei einem diagonalen Sprung $(i, j) \rightarrow (i-1, j-1)$, findet ein Match/Mismatch statt. Es wird der Eintrag $(s[i-1], t[j-1])$ zu A hinzugefügt.
- Bei einem horizontalen Sprung $(i, j) \rightarrow (i, j-1)$, findet eine Insertion statt. Es wird der Eintrag $('-', t[j-1])$ hinzugefügt.
- Bei einem vertikalen Sprung $(i, j) \rightarrow (i-1, j)$, findet eine Deletion statt. Es wird der Eintrag $(s[i-1], '-')$ hinzugefügt.

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit Traceback-Informationen
mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	$(0, -)$	$(1, \leftarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$	$(5, \leftarrow)$
a	$(1, \uparrow)$	$(1, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$
n	$(2, \uparrow)$	$(2, \nwarrow)$	$(2, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$
d	$(3, \uparrow)$	$(3, \nwarrow)$	$(3, \nwarrow)$	$(2, \uparrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$
i	$(4, \uparrow)$	$(4, \nwarrow)$	$(4, \nwarrow)$	$(3, \uparrow)$	$(2, \uparrow)$	$(2, \nwarrow)$

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit Traceback-Informationen
mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	$(0, -)$	$(1, \leftarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$	$(5, \leftarrow)$
a	$(1, \uparrow)$	$(1, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$
n	$(2, \uparrow)$	$(2, \nwarrow)$	$(2, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$
d	$(3, \uparrow)$	$(3, \nwarrow)$	$(3, \nwarrow)$	$(2, \uparrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$
i	$(4, \uparrow)$	$(4, \nwarrow)$	$(4, \nwarrow)$	$(3, \uparrow)$	$(2, \uparrow)$	$(2, \nwarrow)$

$A = []$

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit Traceback-Informationen
mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	$(0, -)$	$(1, \leftarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$	$(5, \leftarrow)$
a	$(1, \uparrow)$	$(1, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$
n	$(2, \uparrow)$	$(2, \nwarrow)$	$(2, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$
d	$(3, \uparrow)$	$(3, \nwarrow)$	$(3, \nwarrow)$	$(2, \uparrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$
i	$(4, \uparrow)$	$(4, \nwarrow)$	$(4, \nwarrow)$	$(3, \uparrow)$	$(2, \uparrow)$	$(2, \nwarrow)$

$$A = \left[\begin{pmatrix} \texttt{i} \\ \texttt{y} \end{pmatrix} \right]$$

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit Traceback-Informationen
mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	$(0, -)$	$(1, \leftarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$	$(5, \leftarrow)$
a	$(1, \uparrow)$	$(1, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$
n	$(2, \uparrow)$	$(2, \nwarrow)$	$(2, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$
d	$(3, \uparrow)$	$(3, \nwarrow)$	$(3, \nwarrow)$	$(2, \uparrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$
i	$(4, \uparrow)$	$(4, \nwarrow)$	$(4, \nwarrow)$	$(3, \uparrow)$	$(2, \uparrow)$	$(2, \nwarrow)$

$$A = \left[\begin{pmatrix} \texttt{d} \\ \texttt{d} \end{pmatrix}, \begin{pmatrix} \texttt{i} \\ \texttt{y} \end{pmatrix} \right]$$

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit Traceback-Informationen
mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	$(0, -)$	$(1, \leftarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$	$(5, \leftarrow)$
a	$(1, \uparrow)$	$(1, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$
n	$(2, \uparrow)$	$(2, \nwarrow)$	$(2, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$
d	$(3, \uparrow)$	$(3, \nwarrow)$	$(3, \nwarrow)$	$(2, \uparrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$
i	$(4, \uparrow)$	$(4, \nwarrow)$	$(4, \nwarrow)$	$(3, \uparrow)$	$(2, \uparrow)$	$(2, \nwarrow)$

$$A = \left[\binom{\texttt{n}}{\texttt{n}}, \binom{\texttt{d}}{\texttt{d}}, \binom{\texttt{i}}{\texttt{y}} \right]$$

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit Traceback-Informationen
mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	$(0, -)$	$(1, \leftarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$	$(5, \leftarrow)$
a	$(1, \uparrow)$	$(1, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$
n	$(2, \uparrow)$	$(2, \nwarrow)$	$(2, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$
d	$(3, \uparrow)$	$(3, \nwarrow)$	$(3, \nwarrow)$	$(2, \uparrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$
i	$(4, \uparrow)$	$(4, \nwarrow)$	$(4, \nwarrow)$	$(3, \uparrow)$	$(2, \uparrow)$	$(2, \nwarrow)$

$$A = \left[\binom{\texttt{a}}{\texttt{a}}, \binom{\texttt{n}}{\texttt{n}}, \binom{\texttt{d}}{\texttt{d}}, \binom{\texttt{i}}{\texttt{y}} \right]$$

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit Traceback-Informationen
mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	$(0, -)$	$(1, \leftarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$	$(5, \leftarrow)$
a	$(1, \uparrow)$	$(1, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$	$(4, \leftarrow)$
n	$(2, \uparrow)$	$(2, \nwarrow)$	$(2, \nwarrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$	$(3, \leftarrow)$
d	$(3, \uparrow)$	$(3, \nwarrow)$	$(3, \nwarrow)$	$(2, \uparrow)$	$(1, \nwarrow)$	$(2, \leftarrow)$
i	$(4, \uparrow)$	$(4, \nwarrow)$	$(4, \nwarrow)$	$(3, \uparrow)$	$(2, \uparrow)$	$(2, \nwarrow)$

$$A = \left[\begin{pmatrix} - \\ \texttt{h} \end{pmatrix}, \begin{pmatrix} \texttt{a} \\ \texttt{a} \end{pmatrix}, \begin{pmatrix} \texttt{n} \\ \texttt{n} \end{pmatrix}, \begin{pmatrix} \texttt{d} \\ \texttt{d} \end{pmatrix}, \begin{pmatrix} \texttt{i} \\ \texttt{y} \end{pmatrix} \right]$$

Implementierung

```
1  def alignment(s, t):
2      m, n = len(s), len(t)
3      Dc = list(range(m + 1))          # current column
4      Dl = [0] * (m + 1)               # last column
5      T = [['-'] * (n + 1) for i in range(m + 1)]
6      for j in range(1, n + 1): T[0][j] = 'i'
7      for i in range(1, m + 1): T[i][0] = 'd'
8
9      for j, tj in zip(range(1, n + 1), t):
10         Dl, Dc = Dc, Dl
11         Dc[0] = j
12         for i, si in zip(range(1, m + 1), s):
13             Dc[i], T[i][j] = min((Dl[i] + 1, 'i'),
14                  (Dl[i - 1] + (si != tj), 'm'),
15                  (Dc[i - 1] + 1, 'd'))
16     return traceback(s, t, T, m, n)
```

Implementierung

```
1  def traceback(s, t, T, i, j):
2      ii, jj, align = 0, 0, []
3      while (ii, jj) != (i, j):
4          ii, jj = i, j
5          if T[i][j] == 'm':
6              i, j = i - 1, j - 1
7              align.append((s[i], t[j]))
8          elif T[i][j] == 'i':
9              j -= 1
10             align.append(("-", t[j]))
11         elif T[i][j] == 'd':
12             i -= 1
13             align.append((s[i], "-"))
14
15     return align[::-1]
```

Globales Alignment mit Scores

- Oftmals kommt es vor, dass die verschiedenen Operationen unterschiedliche Kosten haben.
- Einheitskosten sind somit nicht immer sinnvoll.
- Auch möchte man die verschiedenen Matches unterschiedlich bewerten: $d(\mathtt{a}, \mathtt{c}) < d(\mathtt{a}, \mathtt{g})$.
- Durch die Verwendung von Scores ist dies einfach realisierbar.

Globales Alignment mit Scores

Folgendes muss nun im Algorithmus umgestellt werden:

- Statt Kosten zu minimieren, werden Scores maximiert.
- Eine Scorematrix muss genutzt werden, welche den Score der einzelnen Zeichen miteinander bestimmt. Solch eine Matrix kann auch asymmetrisch sein: $s(\mathtt{a}, \mathtt{c}) = 3$; $s(\mathtt{c}, \mathtt{a}) = 5$. Auch kann der Score einzelner Zeichen von ihrer Position abhängen.
- Auch individuelle Scores für Insertionen / Deletionen können definiert werden. Grundsätzlich können diese Scores symbol- oder positionsabhängig sein. (Bei gleichen In / Del-Scores spricht man von Gap-Kosten.)

Implementierung

```
1  class score:
2      def __init__(self):
3          self.Ins, self.Del = -1, -1
4          self.Score = lambda a, b: -1 + 2 * (a == b)
5
6  def alignment(s, t, sv):  # sv: score-values
7      m, n = len(s), len(t)
8      Sc, Sl = [i * sv.Del for i in range(m + 1)], [0] * (m + 1)
9      T = [['-'] * (n + 1) for i in range(m + 1)]
10     for j in range(1, n + 1): T[0][j] = 'i'
11     for i in range(1, m + 1): T[i][0] = 'd'
12     for j, tj in zip(range(1, n + 1), t):
13         Sl, Sc = Sc, Sl
14         Sc[0] = j * sv.Ins
15         for i, si in zip(range(1, m + 1), s):
16             Sc[i], T[i][j] = max((Sl[i] + sv.Ins, 'i'),
17                 (Sl[i - 1] + sv.Score(si, tj), 'm'),
18                 (Sc[i - 1] + sv.Del, 'd'))
19     return traceback(s, t, T, m, n)
```

Universeller Alignment-Algorithmus

Um von allen Gegebenheiten unabhängig zu sein, kann wieder der Alignment-Graph genutzt werden mit folgenden Eigenschaften:

- Sei $S(v)$ der maximale Score aller Pfade von $v_\circ$ nach v.
- Sei $T(v)$ der Vorgängerknoten, über dessen Pfad der maximale Score geht.
- Sei $S(v_\circ) = 0$.

Universeller Alignment-Algorithmus

Um von allen Gegebenheiten unabhängig zu sein, kann wieder der Alignment-Graph genutzt werden mit folgenden Eigenschaften:

- Sei $S(v)$ der maximale Score aller Pfade von $v_\circ$ nach v.
- Sei $T(v)$ der Vorgängerknoten, über dessen Pfad der maximale Score geht.
- Sei $S(v_\circ) = 0$.
- Für alle $v \neq v_\circ$ in topologisch sortierter Reihenfolge gilt:

$$S(v) = \max_{w:\, w \to v \in E} \{S(w) + \text{score}(w \to v)\},$$

$$T(v) = \arg\max_{w:\, w \to v \in E} \{S(w) + \text{score}(w \to v)\}.$$

Universeller Alignment-Algorithmus

- Durch die Berechnung in topologisch sortierter Reihenfolge ist sichergestellt, dass $S(w)$ bereits bekannt ist, wenn es bei der Berechnung von $S(v)$ ausgewertet werden soll.
- Offensichtlich ist dazu notwendig, dass der Alignment-Graph keine gerichteten Zyklen enthält.
- Den optimalen Score erhalten wir als $S(v_\bullet)$.
- Den optimalen Pfad erhalten wir durch Traceback von $v_\bullet$ aus, indem wir jeweils zum durch T gegebenen Vorgängerknoten gehen:
  $v_\bullet \to T(v_\bullet) \to T(T(v_\bullet)) \to \cdots \to T^k(v_\bullet) \to \cdots \to v_\circ$.

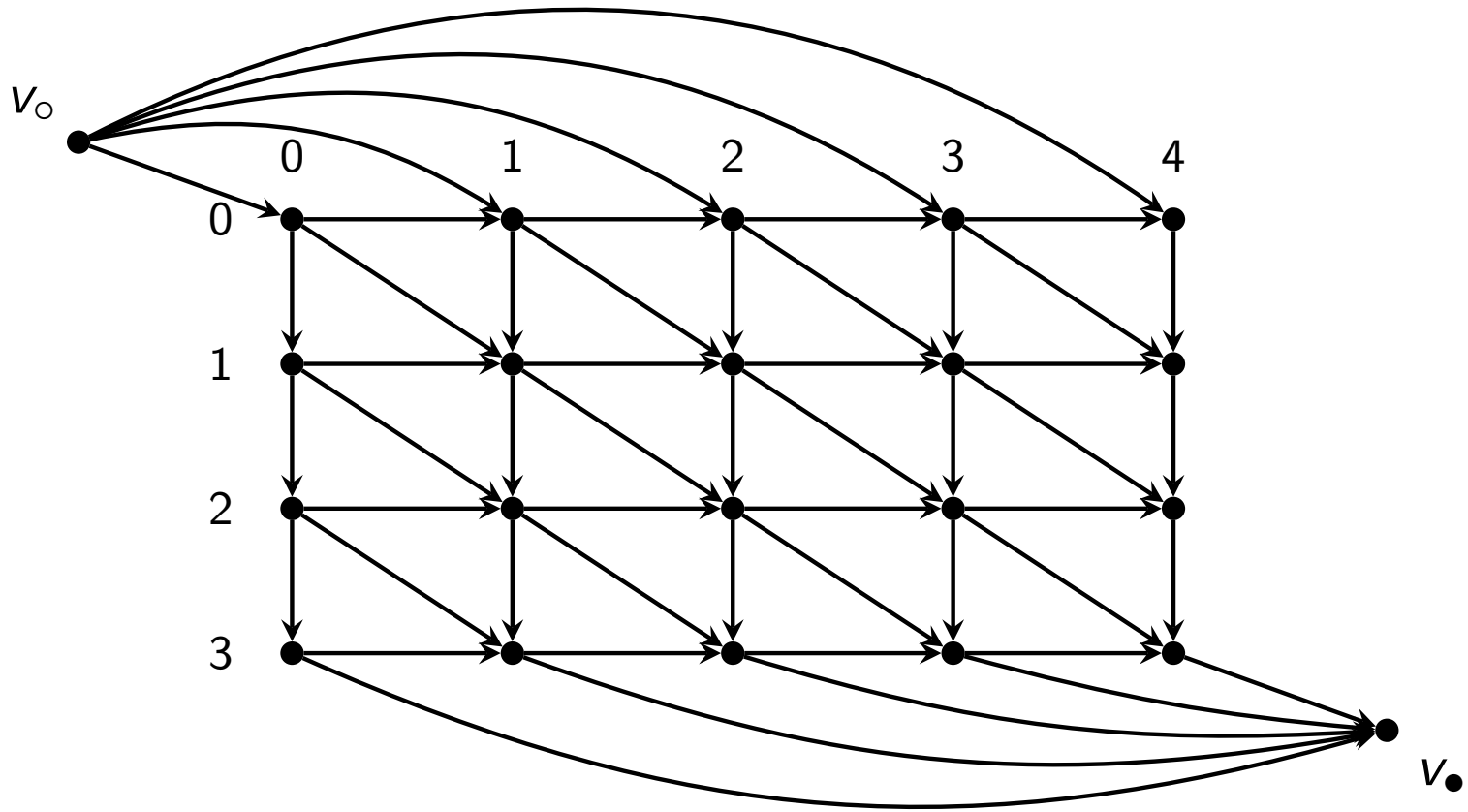
Verschiedene Alignment-Varianten

Semiglobales Alignment:

- Wurde bereits im letzten Abschnitt eingeführt.
- Wenn keine Einheitskosten vorausgesetzt sind, können die Optimierungen (Ukkonen's Algorithmus, bit-parallele NFA-Simulation, fehlertolerante Backward Search) nicht mehr ohne Weiteres angewendet werden.
- Tatsächlich lässt sich aber mit dem für semiglobales Alignment modifizierten Graphen der beste Score bestimmen.
- Es gibt zusätzliche Initialisierungskanten $(v_\circ) \to (0, j)$ und Finalisierungskanten $(m, j) \to (v_\bullet)$ für alle j.

Semiglobales Alignment

Beispiel für einen Alignment-Graph für ein semiglobales Alignment:



Verschiedene Alignment-Varianten

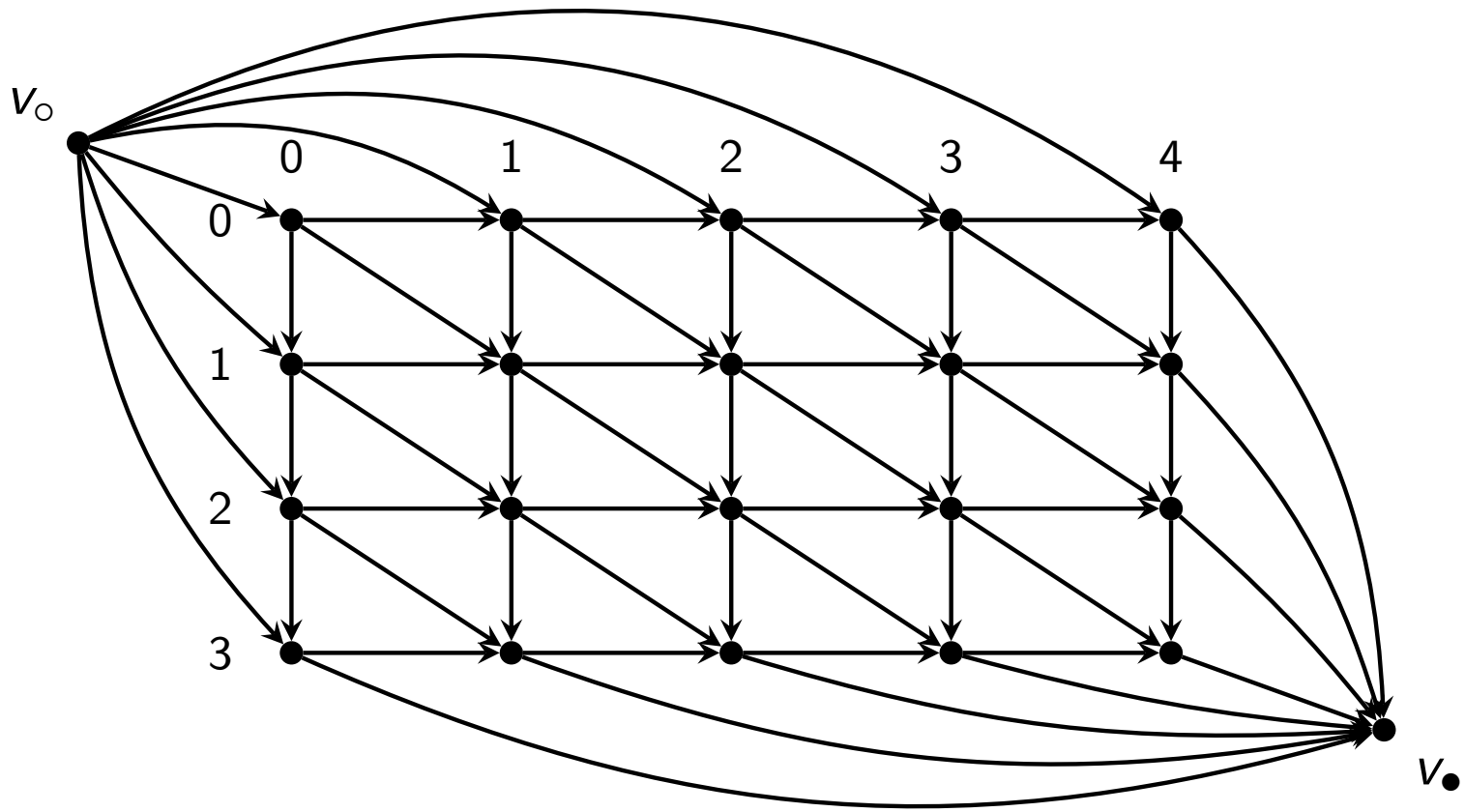
"Free End Gaps"-Alignment:

- Fragestellung: überlappen sich zwei Sequenzen?

- Anfang bzw. Ende dürfen daher nicht bestraft werden.
- Es gibt zusätzliche kostenfreie Initialisierungskanten $(v_\circ) \to (i, 0), (v_\circ) \to (0, j)$ und Finalisierungskanten $(i, n) \to (v_\bullet), (m, j) \to (v_\bullet)$.
- Mit kostenfrei ist ein Score mit Wert 0 gemeint.

“Free End Gaps”-Alignment

Beispiel für einen Alignment-Graph für ein “Free End Gaps”-Alignment:



Verschiedene Alignment-Varianten

Lokales Alignment:

- Gesucht ist das größte optimale Alignment zwischen allen Teilstrings s' (von s) und t' (von t).

- Beim naïven Ansatz werden alle Paare von Teilstrings miteinander verglichen, die Laufzeit beträgt $\mathcal{O}(m^2 n^2)$.

Verschiedene Alignment-Varianten

Lokales Alignment:

- Gesucht ist das größte optimale Alignment zwischen allen Teilstrings s' (von s) und t' (von t).

- Beim naïven Ansatz werden alle Paare von Teilstrings miteinander verglichen, die Laufzeit beträgt $\mathcal{O}(m^2 n^2)$.
- Wieder kann der universelle Alignment-Graph für dieses Problem angepasst werden:
  - Initialisierungskanten: $(v_\circ) \to (i, j)$ für alle $i \leq m, j \leq n$,
  - Finalisierungskanten: $(i, j) \to (v_\bullet)$ für alle $i \leq m, j \leq n$.

Verschiedene Alignment-Varianten

Lokales Alignment:

- Die Initialisierungsspalten/zeilen müssen angepasst werden.
- Da man von jedem Knoten starten kann, muss bei der Scoreberechnung noch ein 0-Score hinzugefügt werden.
- Pro Berechnung eines Feldes muss überprüft (und ggf. gespeichert) werden, ob der neue Score größer als der bisher größte Score ist, da man an jedem Knoten enden kann.

Verschiedene Alignment-Varianten

Lokales Alignment:

- Die Initialisierungsspalten/zeilen müssen angepasst werden.
- Da man von jedem Knoten starten kann, muss bei der Scoreberechnung noch ein 0-Score hinzugefügt werden.
- Pro Berechnung eines Feldes muss überprüft (und ggf. gespeichert) werden, ob der neue Score größer als der bisher größte Score ist, da man an jedem Knoten enden kann.
- Die Laufzeit sinkt somit auf $\mathcal{O}(mn)$.

Lokales Alignment

```
1  def local_alignment(s, t, sv): # sv: score-values
2      m, n, best, b_i, b_j = len(s), len(t), 0, 0, 0
3      Sc, Sl = [0] * (m + 1), [0] * (m + 1)
4      T = [['-'] * (n + 1) for i in range(m + 1)]
5
6      for j, tj in zip(range(1, n + 1), t):
7          Sl, Sc = Sc, Sl
8          for i, si in zip(range(1, m + 1), s):
9              Sc[i], T[i][j] = max((Sl[i] + sv.Ins, 'i'),
10                 (Sl[i - 1] + sv.Score(si, tj), 'm'),
11                 (Sc[i - 1] + sv.Del, 'd'),
12                 (0, '-'))
13             if best < Sc[i]: best, b_i, b_j = Sc[i], i, j
14     return best, traceback(s, t, T, b_i, b_j)
```

Allgemeine Gapkosten

- Im Folgenden werden InDels gleichermaßen als Gaps betrachtet.
- Bisher hat ein Gap der Länge ℓ einen Score von $g(\ell) = -\gamma \cdot \ell$, wobei $\gamma \geq 0$ die Gapkosten sind (negativer Score eines Gaps).
- Diese Annahme ist z.B. bei biologischen Sequenzen nicht sonderlich realistisch.

```
attccgacagaaagatac      attccgacagaaagatac
att-c--c-g----atac  >   attccg--------atac
```

Allgemeine Gapkosten

- Im Folgenden werden InDels gleichermaßen als Gaps betrachtet.
- Bisher hat ein Gap der Länge ℓ einen Score von $g(\ell) = -\gamma \cdot \ell$, wobei $\gamma \geq 0$ die Gapkosten sind (negativer Score eines Gaps).
- Diese Annahme ist z.B. bei biologischen Sequenzen nicht sonderlich realistisch.

```
attccgacagaaagatac      attccgacagaaagatac
att-c--c-g----atac   >  attccg--------atac
```

- Besser ist es, die Kosten für das Eröffnen eines Gaps höher anzusetzen als die für das Verlängern eines Gaps.

Allgemeine Gapkosten

- Eine relativ einfache, aber schon relativ realistische Modellierung erreichen wir durch *affine Gapkosten*

$$g(\ell) := -c - \gamma(\ell - 1) \text{ für } \ell \geq 1.$$

- Dabei gibt die Konstante c die Gapöffnungskosten (gap open penalty) und γ die Gaperweiterungskosten (gap extension penalty) an.

Globales Alignment mit affinen Gapkosten

- Ziel ist es einen Algorithmus mit einer Laufzeit von $\mathcal{O}(mn)$ zu entwickeln.
- Seien dabei c die *gap open penalty* und γ die *gap extension penalty*.
- Ferner soll gelten: $c \leq \gamma \leq 0$.
- Die Knoten im Alignment-Graphen sollen nun drei Informationen speichern.

Globales Alignment mit affinen Gapkosten

- $S[i][j] := \max\left\{ score(A) \mid \quad A \text{ ist ein Alignment von } s[:i] \text{ und } t[:j] \quad \right\}.$
- Zusätzlich soll gespeichert werden:
- $V[i][j] := \max\left\{ score(A) \mid \begin{array}{l} A \text{ ist ein Alignment von } s[:i] \text{ und } t[:j] \\ \text{das mit einem Gap } (-) \text{ in } t \text{ endet} \end{array} \right\},$
- $H[i][j] := \max\left\{ score(A) \mid \begin{array}{l} A \text{ ist ein Alignment von } s[:i] \text{ und } t[:j] \\ \text{das mit einem Gap } (-) \text{ in } s \text{ endet} \end{array} \right\}.$

Globales Alignment mit affinen Gapkosten

Daraus ergeben sich folgende Rekurrenzen:

$$V[i][j] = \max\{S[i-1][j] - c,\ V[i-1][j] - \gamma\},$$
$$H[i][j] = \max\{S[i][j-1] - c,\ H[i][j-1] - \gamma\},$$
$$S[i][j] = \max\{S[i-1][j-1] + \text{score}(s[i-1], t[j-1]),\ V[i][j],\ H[i][j]\}.$$

Globales Alignment mit affinen Gapkosten

Wichtig ist eine korrekte Initialisierung der Tabellen, sei hierzu:

$$
\begin{aligned}
S[0][0] &= 0, \\
V[0][j] &= -\infty \text{ für alle } 0 \leq j \leq n, \\
S[i][0] &= V[i][0] = g(i) \text{ für alle } 1 \leq i \leq m, \\
H[i][0] &= -\infty \text{ für alle } 0 \leq i \leq m, \\
S[0][j] &= H[0][j] = g(j) \text{ für alle } 1 \leq j \leq m.
\end{aligned}
$$

Globales Alignment mit affinen Gapkosten

Wichtig ist eine korrekte Initialisierung der Tabellen, sei hierzu:

$$
\begin{aligned}
S[0][0] &= 0, \\
V[0][j] &= -\infty \text{ für alle } 0 \leq j \leq n, \\
S[i][0] &= V[i][0] = g(i) \text{ für alle } 1 \leq i \leq m, \\
H[i][0] &= -\infty \text{ für alle } 0 \leq i \leq m, \\
S[0][j] &= H[0][j] = g(j) \text{ für alle } 1 \leq j \leq m.
\end{aligned}
$$

Achtung: Die herkömmliche Traceback-Funktion darf nicht mehr genutzt werden, da ein Match und Gapverlängerung den selben Score bekommen können.

Alignments mit Einschränkungen

- Wie können Alignments gefunden werden, wenn ein (oder mehrere) Punkt(e) vorgegeben ist/sind.
- Gesucht ist also ein Pfad, der durch den Knoten (i, j) führt.

Alignments mit Einschränkungen

- Wie können Alignments gefunden werden, wenn ein (oder mehrere) Punkt(e) vorgegeben ist/sind.
- Gesucht ist also ein Pfad, der durch den Knoten (i, j) führt.
- Lösung: zwei Alignments berechnen:

$$(0, 0) \rightarrow (i, j) \text{ und } (i, j) \rightarrow (m, n).$$

Alignments mit Einschränkungen

- Schwieriger ist es für alle Knoten (i, j) *gleichzeitig* ein optimales Alignment zu berechnen.
- Ein naïver Ansatz wäre für jeden Knoten (i, j) die beiden Alignments zu berechnen, die Laufzeit beträgt $\mathcal{O}(m^2 n^2)$.

Alignments mit Einschränkungen

- Schwieriger ist es für alle Knoten (i, j) *gleichzeitig* ein optimales Alignment zu berechnen.
- Ein naïver Ansatz wäre für jeden Knoten (i, j) die beiden Alignments zu berechnen, die Laufzeit beträgt $\mathcal{O}(m^2 n^2)$.
- Eine Beobachtung ist aber, dass der Score des optimalen Pfades von $(0, 0)$ nach (i, j) für alle (i, j) bereits in der DP-Tabelle als $S[i][j]$ gespeichert ist.
- Die Lösung „der ersten Hälfte“ des Problems kann einfach abgelesen werden.

Alignments mit Einschränkungen

- Schwieriger ist es für alle Knoten (i, j) *gleichzeitig* ein optimales Alignment zu berechnen.
- Ein naïver Ansatz wäre für jeden Knoten (i, j) die beiden Alignments zu berechnen, die Laufzeit beträgt $\mathcal{O}(m^2 n^2)$.
- Eine Beobachtung ist aber, dass der Score des optimalen Pfades von $(0, 0)$ nach (i, j) für alle (i, j) bereits in der DP-Tabelle als $S[i][j]$ gespeichert ist.
- Die Lösung „der ersten Hälfte“ des Problems kann einfach abgelesen werden.
- Wie kann „der zweite Hälfte“ gelöst werden?

Alignments mit Einschränkungen

- Trick: Um auch den optimalen Score von (i, j) nach (m, n) einfach ablesen zu können, muss der Algorithmus nur rückwärts ausgeführt werden.
- Auf diese Weise erhält man eine zweite Matrix $R[i][j]$ mit den optimales Scores von $(m, n) \rightarrow (i, j)$.

Alignments mit Einschränkungen

- Trick: Um auch den optimalen Score von (i, j) nach (m, n) einfach ablesen zu können, muss der Algorithmus nur rückwärts ausgeführt werden.
- Auf diese Weise erhält man eine zweite Matrix $R[i][j]$ mit den optimales Scores von $(m, n) \rightarrow (i, j)$.
- Es muss auf die korrekte Indizierung geachtet werden.
- Durch Aufsummierung von $S + R$ erhält mal den optimalen Score aller Pfade, die durch (i, j) laufen.

Alignments mit Einschränkungen

Beispiel mit $s =$ `andi` und $t =$ `handy`.

	ϵ	h	a	n	d	y
ϵ	0	-1	-2	-3	-4	-5
a	-1	-1	0	-1	-2	-3
n	-2	-2	-1	1	0	-1
d	-3	-3	-2	0	2	1
i	-4	-4	-3	-1	1	1

1	2	0	-2	-4	-4	a
-1	0	1	-1	-3	-3	n
-3	-2	-1	0	-2	-2	d
-5	-4	-3	-2	-1	-1	i
-5	-4	-3	-2	-1	0	ϵ
h	a	n	d	y	ϵ	

Alignments mit Einschränkungen

Beispiel mit $s =$ andi und $t =$ handy.

	ϵ	h	a	n	d	y
ϵ	0	-1	-2	-3	-4	-5
a	-1	-1	0	-1	-2	-3
n	-2	-2	-1	1	0	-1
d	-3	-3	-2	0	2	1
i	-4	-4	-3	-1	1	1

1	2	0	-2	-4	-4	a
-1	0	1	-1	-3	-3	n
-3	-2	-1	0	-2	-2	d
-5	-4	-3	-2	-1	-1	i
-5	-4	-3	-2	-1	0	ϵ
h	a	n	d	y	ϵ	

$\searrow$ + $\swarrow$

1	1	-2	-5	-8	-9
-2	-1	1	-2	-5	-6
-5	-4	-2	1	-2	-3
-8	-7	-5	-2	1	0
-9	-8	-6	-3	0	1

Alignment mit linearem Platzbedarf

- Bisher war es nur möglich die Edit-Distanz (optimalen Score) mit linearem Platzbedarf in $\mathcal{O}(\min(m, n))$ zu berechnen.
- Das herkömmliche Traceback-Verfahren braucht $\mathcal{O}(mn)$ Speicherplatz.
- Es ist bemerkenswert, dass mit einem Divide-and-Conquer Ansatz, der Speicherplatz auf $\mathcal{O}(m + n)$ reduziert werden kann, ohne dass die Asymptotische Laufzeit von $\mathcal{O}(mn)$ zunimmt.

Alignment mit linearem Platzbedarf

- Bisher war es nur möglich die Edit-Distanz (optimalen Score) mit linearem Platzbedarf in $\mathcal{O}(\min(m, n))$ zu berechnen.
- Das herkömmliche Traceback-Verfahren braucht $\mathcal{O}(mn)$ Speicherplatz.
- Es ist bemerkenswert, dass mit einem Divide-and-Conquer Ansatz, der Speicherplatz auf $\mathcal{O}(m + n)$ reduziert werden kann, ohne dass die Asymptotische Laufzeit von $\mathcal{O}(mn)$ zunimmt.
- Das Verfahren wurde nach seinem Entwickler *Hirschberg* (1975) benannt.

Hirschberg-Algorithmus

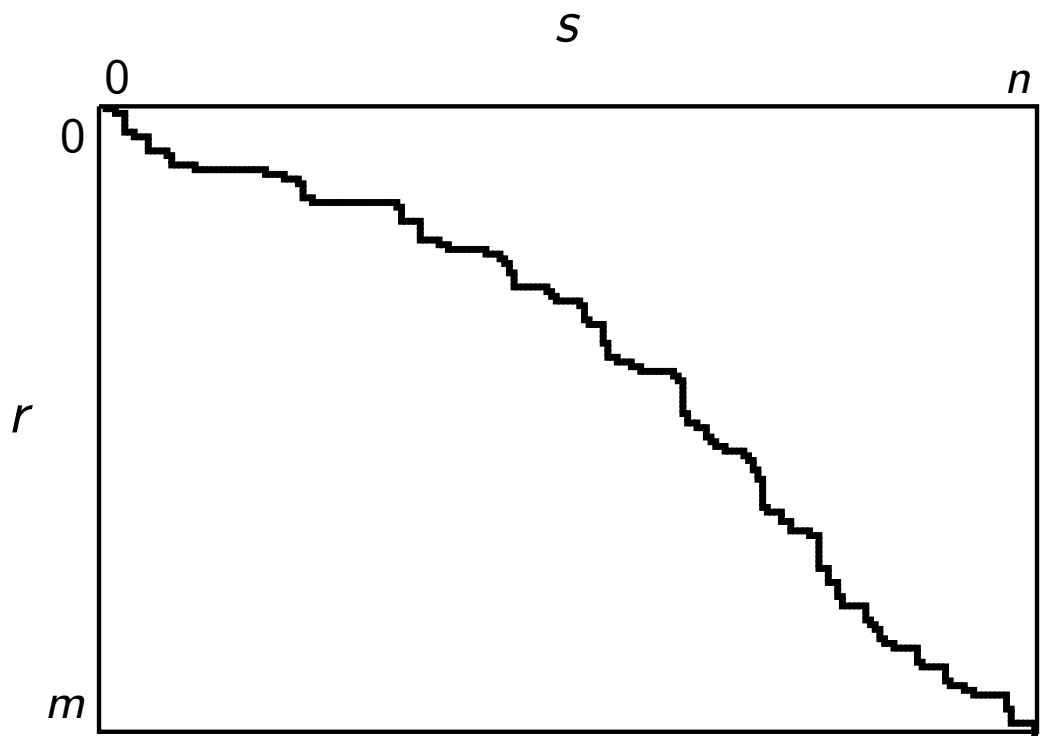
- Es wird die Idee vom vorherigen Abschnitt aufgegriffen, dass durch ein Vorwärts- und Rückwärtsalignment der optimale Score des Pfads, der durch (i, j) verläuft, berechnet werden kann.
- Interessant ist, durch welchen Knoten der optimale Pfad verläuft, wenn die Matrix durch einen vertikalen Schnitt in der Hälfte geteilt wird.

Hirschberg-Algorithmus

- Es wird die Idee vom vorherigen Abschnitt aufgegriffen, dass durch ein Vorwärts- und Rückwärtsalignment der optimale Score des Pfads, der durch (i, j) verläuft, berechnet werden kann.
- Interessant ist, durch welchen Knoten der optimale Pfad verläuft, wenn die Matrix durch einen vertikalen Schnitt in der Hälfte geteilt wird.
- Mit diesem Wissen, kann rekursiv der optimale Pfad ermittelt werden.

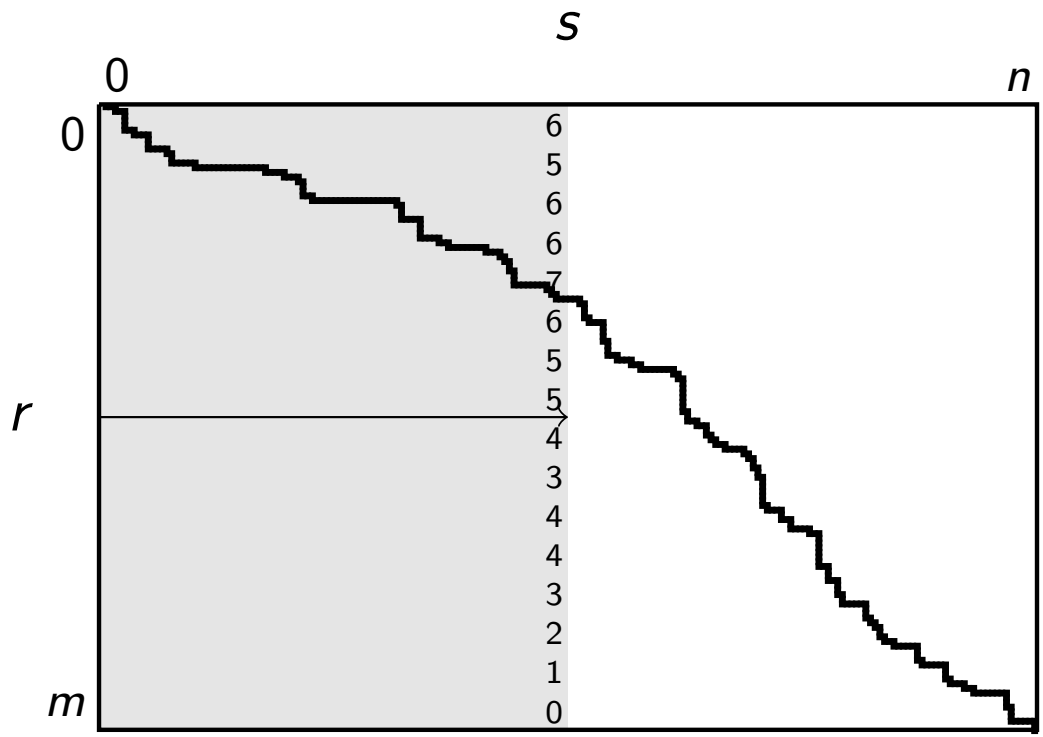
Hirschberg-Algorithmus

Zerlegung des Problems in zwei Teilprobleme:



Hirschberg-Algorithmus

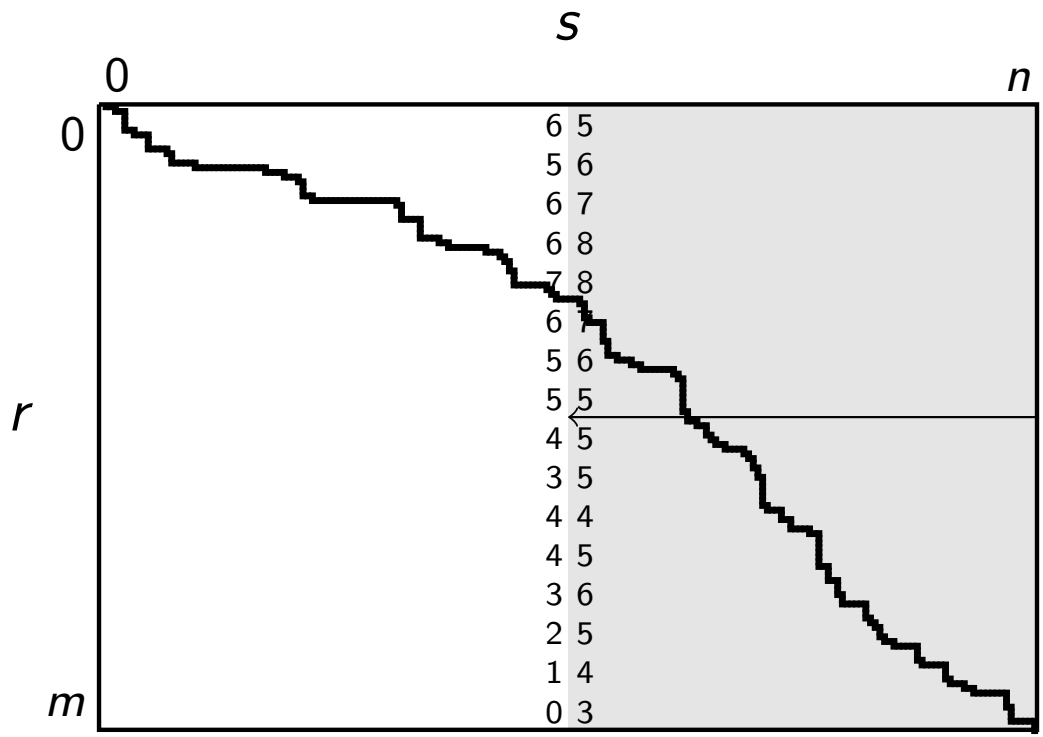
Zerlegung des Problems in zwei Teilprobleme:



Mit dem Vorwärtsalignment von 0 bis $n/2$ alignieren

Hirschberg-Algorithmus

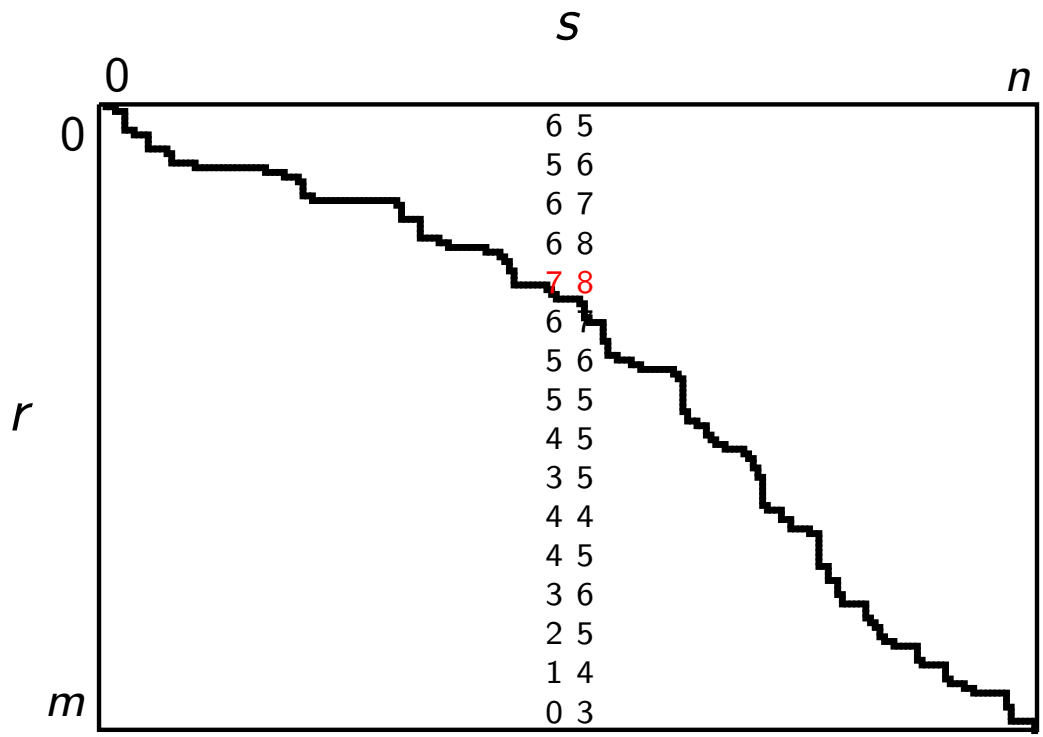
Zerlegung des Problems in zwei Teilprobleme:



Mit dem Rückwärtsalignment von n bis $n/2$ alignieren

Hirschberg-Algorithmus

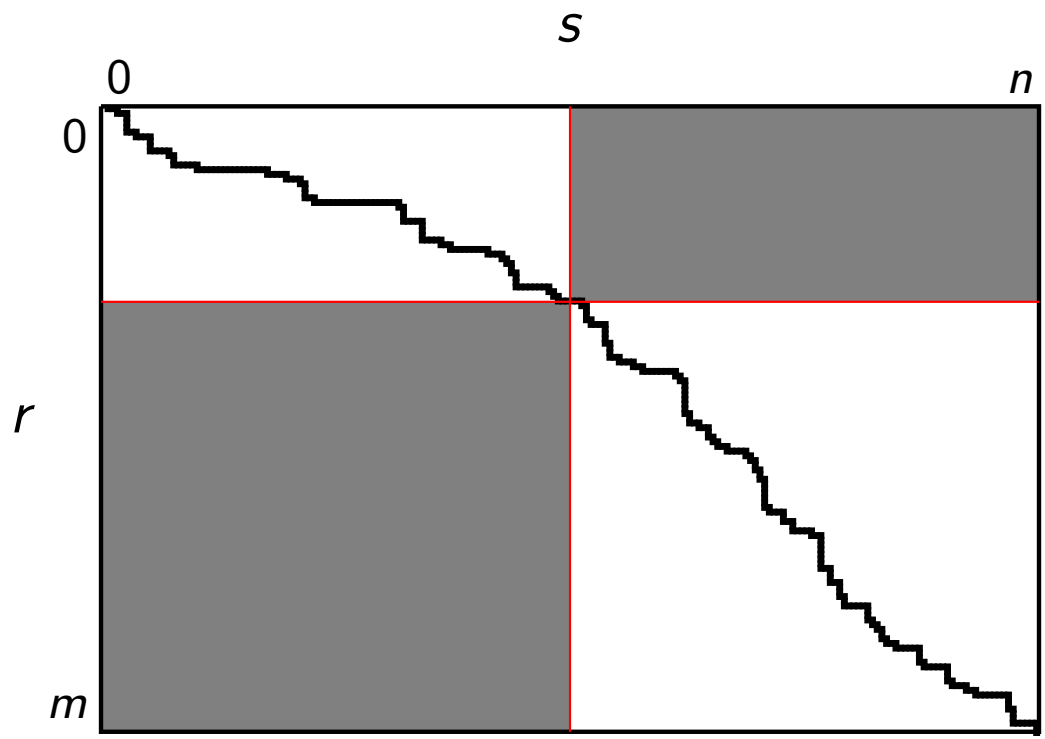
Zerlegung des Problems in zwei Teilprobleme:



Knoten $(i^*, n/2)$ mit dem optimalen Score bestimmen

Hirschberg-Algorithmus

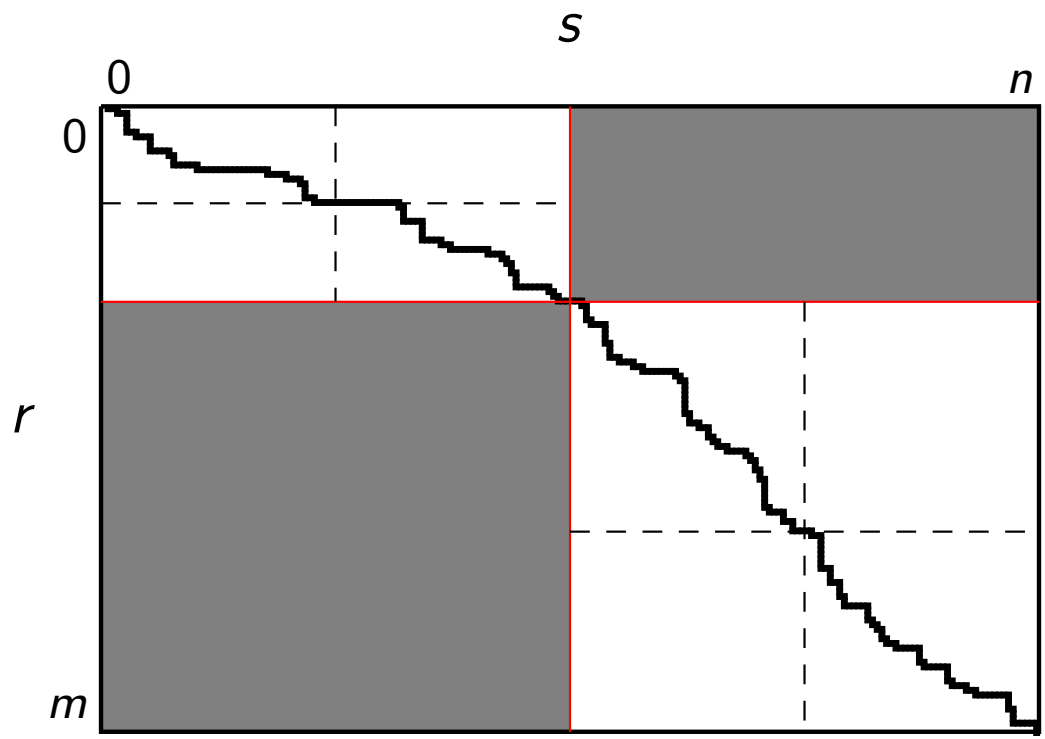
Zerlegung des Problems in zwei Teilprobleme:



Alignments $(0,0) \rightarrow (i^*, n/2)$ und $(i^*, n/2) \rightarrow (m,n)$ durchführen

Hirschberg-Algorithmus

Zerlegung des Problems in zwei Teilprobleme:



Alignments so lange teilen, bis $r[i : i']$ oder $s[j : j']$ Größe 1 hat

Hirschberg-Algorithmus

- Sobald ein Alignment $(i, j) \to (i', j')$ mit $i = i'$ oder $j + 1 = j'$ erreicht wird, wird die Rekursion abgebrochen und mittels Traceback ein herkömmliches Alignment berechnet.
- Da nach jedem Rekursionsschritt die Scorespalten nicht mehr benötigt werden, hängt der benötigte Speicherplatz von der größten Spalte (0. Iteration) ab, also $\mathcal{O}(\min(m, n))$.

Hirschberg-Algorithmus

- Sobald ein Alignment $(i, j) \to (i', j')$ mit $i = i'$ oder $j + 1 = j'$ erreicht wird, wird die Rekursion abgebrochen und mittels Traceback ein herkömmliches Alignment berechnet.
- Da nach jedem Rekursionsschritt die Scorespalten nicht mehr benötigt werden, hängt der benötigte Speicherplatz von der größten Spalte (0. Iteration) ab, also $\mathcal{O}(\min(m, n))$.
- Das Alignment wird beim Rekursionsanker zurückgegeben, verbraucht also insgesamt $\mathcal{O}(m + n)$ Speicher.
- Der gesamte zusätzliche Speicher beträgt also $\mathcal{O}(m + n)$.

Hirschberg-Algorithmus

Beispiel mit $r = \texttt{GAG}$ und $s = \texttt{CACG}$:

	ϵ	C	A	C	G
ϵ	0	−1	−2	−3	−4
G	−1	−1	−2	−3	−2
A	−2	−2	0	−1	−2
G	−3	−3	−1	−1	0

Hirschberg-Algorithmus

Beispiel mit $r = \texttt{GAG}$ und $s = \texttt{CACG}$:

	ϵ	C	A	C	G
ϵ	0	−1	−2	−3	−4
G	−1	−1	−2	−3	−2
A	−2	−2	0	−1	−2
G	−3	−3	−1	−1	0

Hirschberg-Algorithmus

Beispiel mit $r = \texttt{GAG}$ und $s = \texttt{CACG}$:

	ϵ	C	A	C	G
ϵ	0	−1	−2	−3	−4
G	−1	−1	−2	−3	−2
A	−2	−2	0	−1	−2
G	−3	−3	−1	−1	0

Hirschberg-Algorithmus

Beispiel mit $r = \texttt{GAG}$ und $s = \texttt{CACG}$:

	ϵ	C	A	C	G
ϵ	0	−1	−2	−3	−4
G	−1	−1	−2	−3	−2
A	−2	−2	0	−1	−2
G	−3	−3	−1	−1	0

align
GC mit CA

	ϵ	C	A
ϵ	0	−1	−2
G	−1	−1	−2
A	−2	−2	0

Hirschberg-Algorithmus

Beispiel mit $r = \texttt{GAG}$ und $s = \texttt{CACG}$:

	ϵ	C	A	C	G
ϵ	0	−1	−2	−3	−4
G	−1	−1	−2	−3	−2
A	−2	−2	0	−1	−2
G	−3	−3	−1	−1	0

align
GC mit CA

	ϵ	C	A
ϵ	0	−1	−2
G	−1	−1	−2
A	−2	−2	0

Hirschberg-Algorithmus

Beispiel mit $r = \texttt{GAG}$ und $s = \texttt{CACG}$:

	ϵ	C	A	C	G
ϵ	0	−1	−2	−3	−4
G	−1	−1	−2	−3	−2
A	−2	−2	0	−1	−2
G	−3	−3	−1	−1	0

align
GC mit CA

	ϵ	C	A
ϵ	0	−1	−2
G	−1	−1	−2
A	−2	−2	0

Hirschberg-Algorithmus

Beispiel mit $r = \mathtt{GAG}$ und $s = \mathtt{CACG}$:

	ϵ	C	A	C	G
ϵ	0	−1	−2	−3	−4
G	−1	−1	−2	−3	−2
A	−2	−2	0	−1	−2
G	−3	−3	−1	−1	0

align GC mit CA

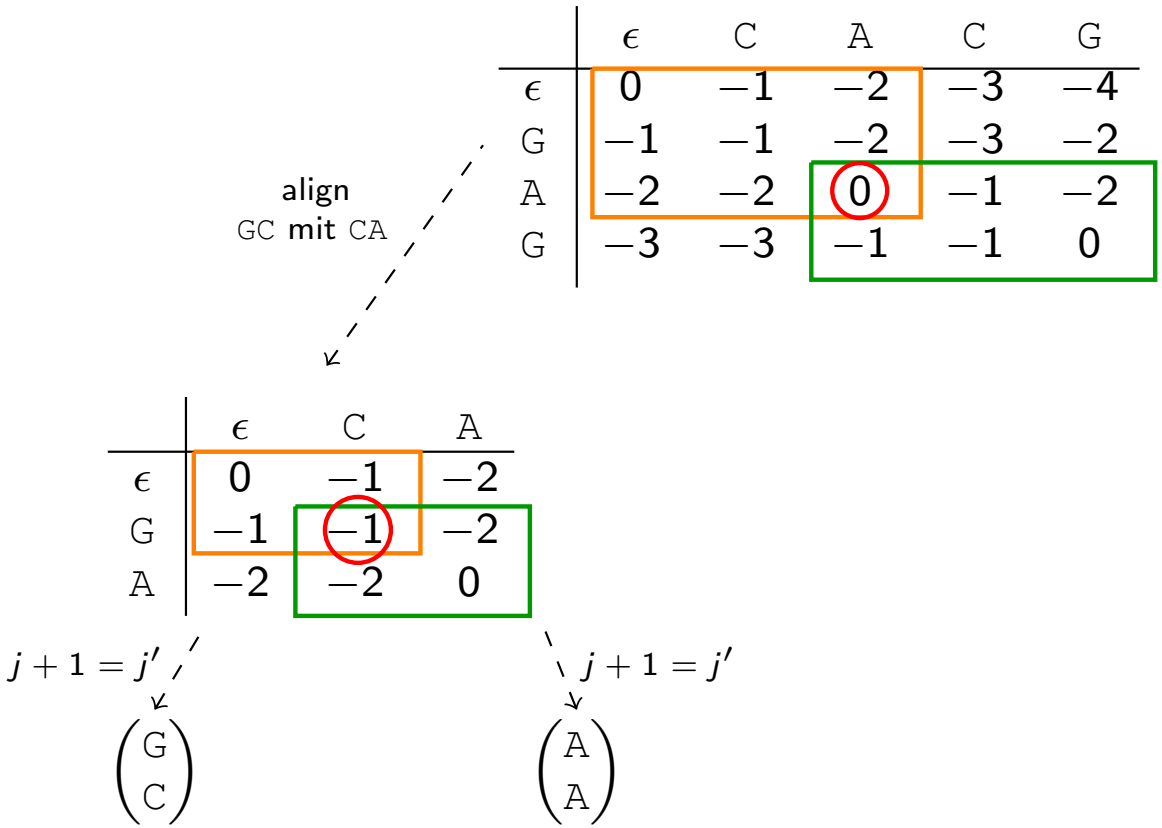
	ϵ	C	A
ϵ	0	−1	−2
G	−1	−1	−2
A	−2	−2	0

$j + 1 = j'$

$\begin{pmatrix} \mathtt{G} \\ \mathtt{C} \end{pmatrix}$

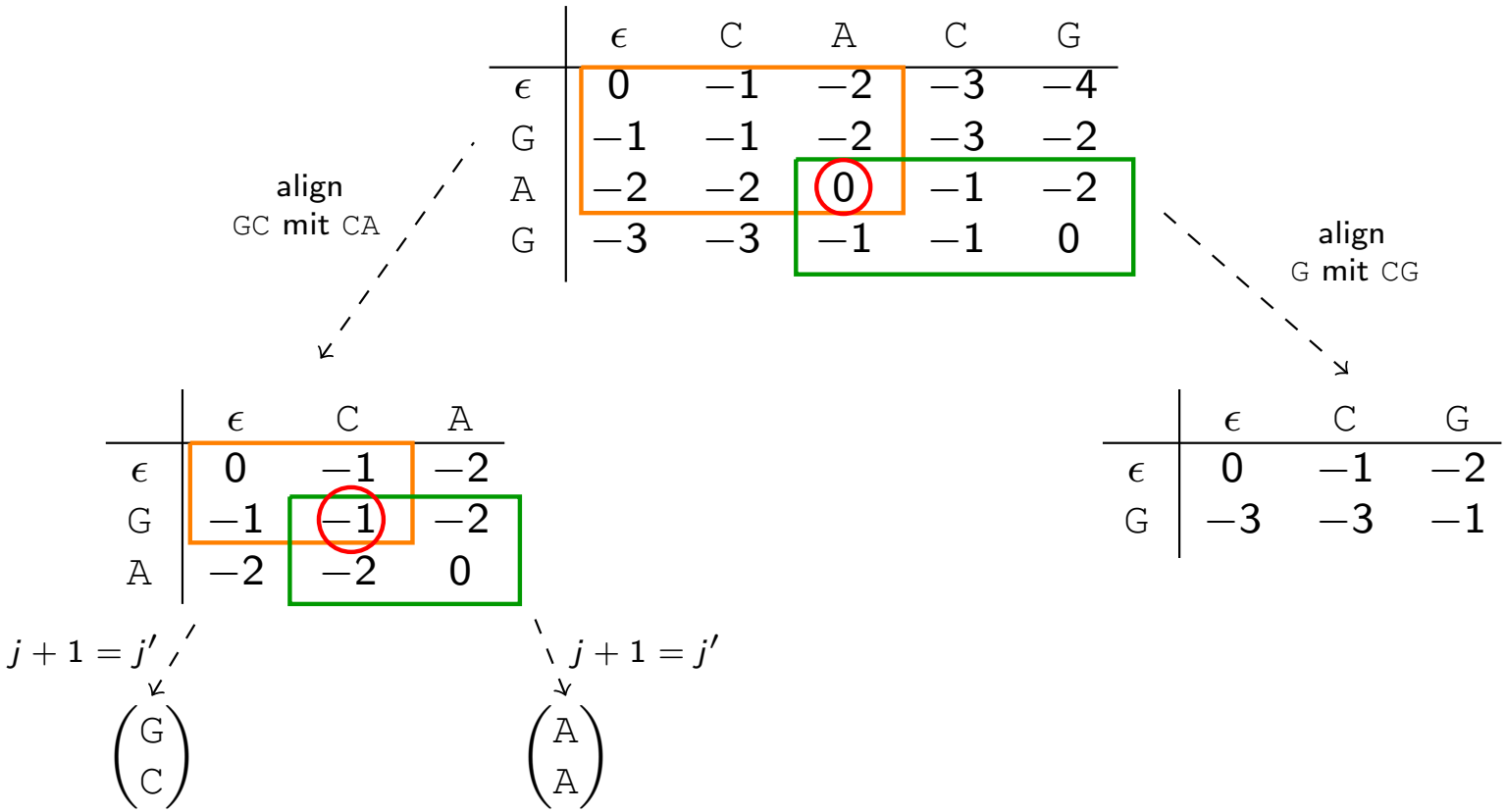
Hirschberg-Algorithmus

Beispiel mit $r = \mathtt{GAG}$ und $s = \mathtt{CACG}$:



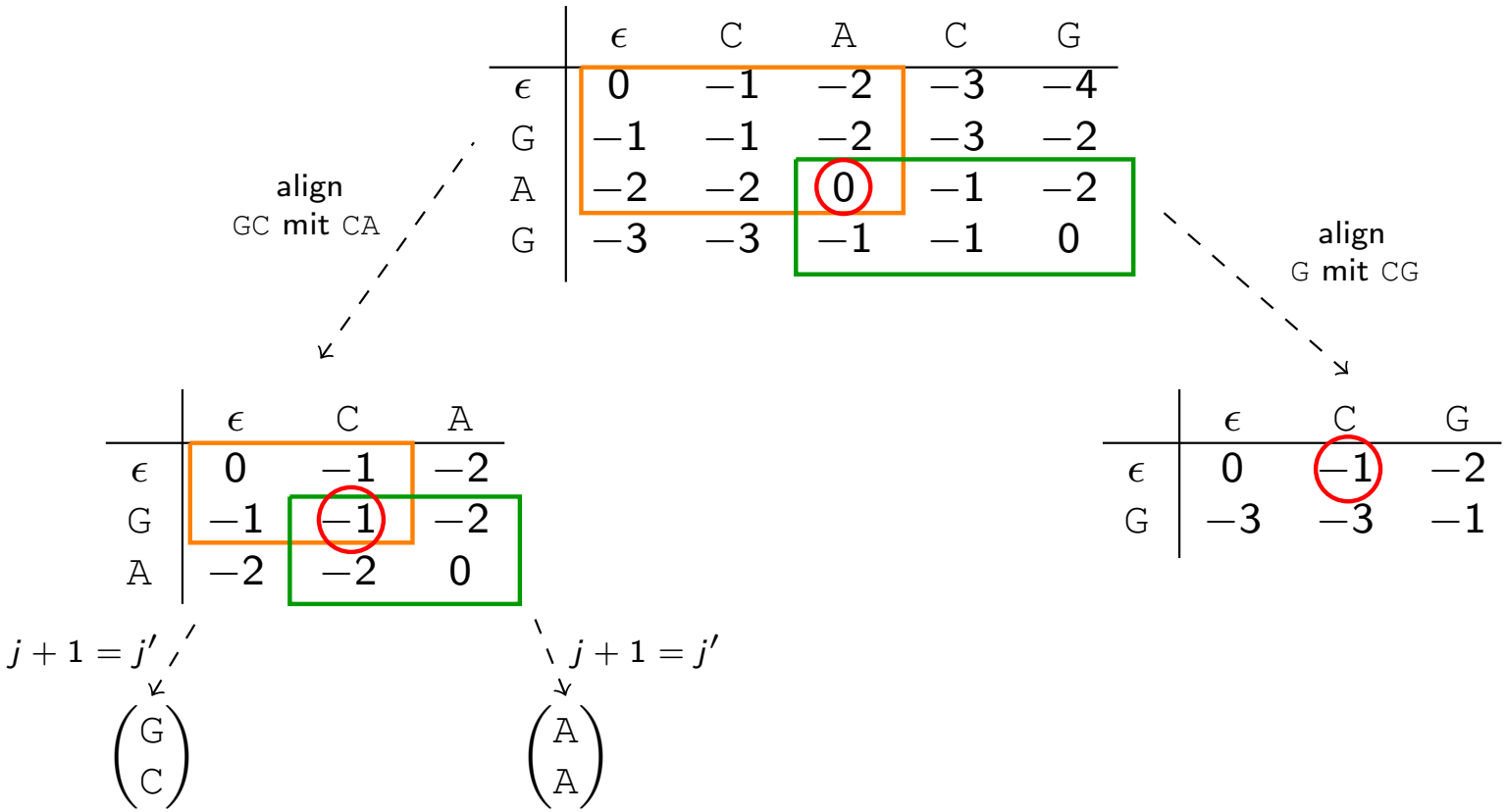
Hirschberg-Algorithmus

Beispiel mit $r = \mathtt{GAG}$ und $s = \mathtt{CACG}$:



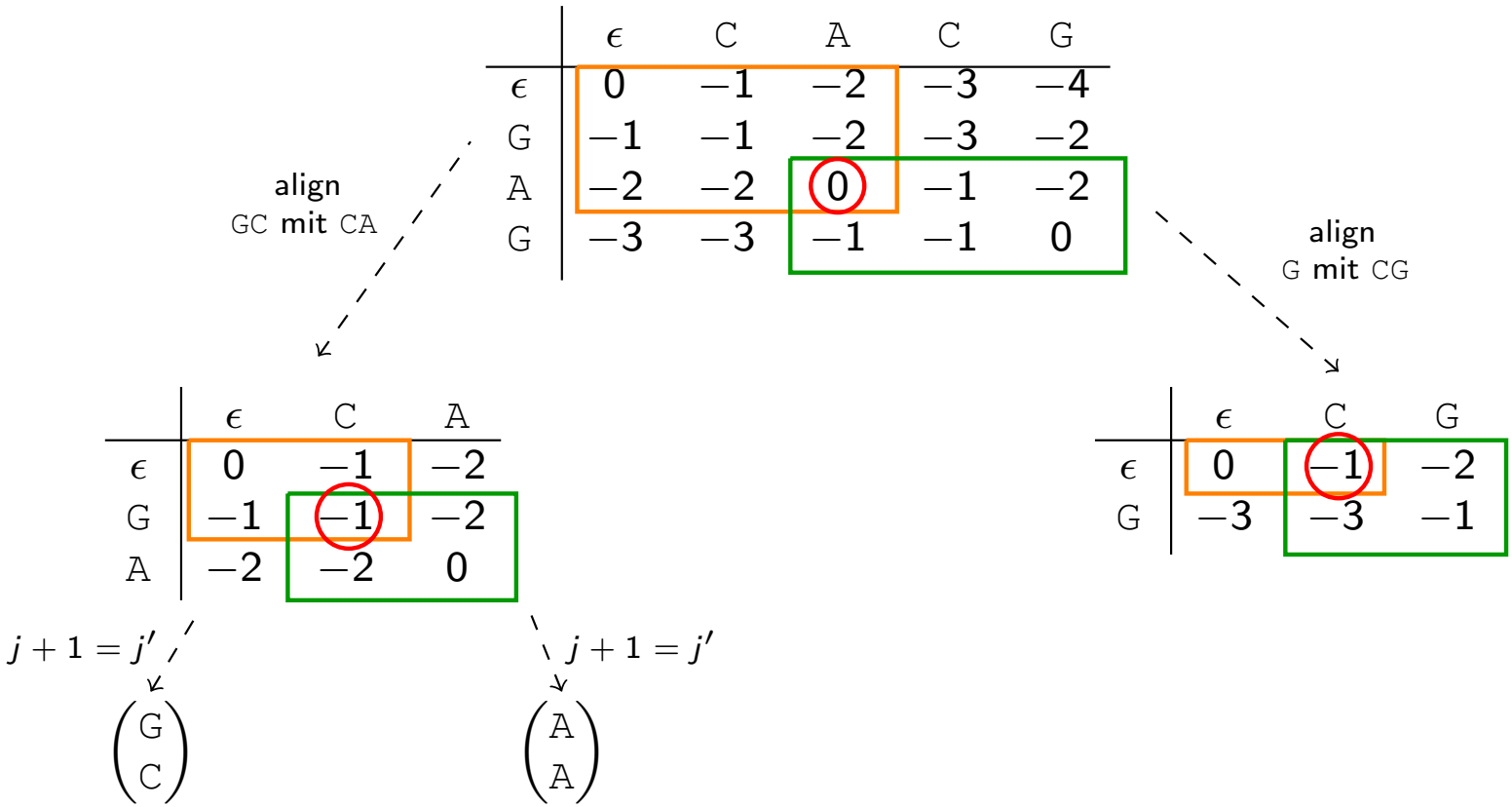
Hirschberg-Algorithmus

Beispiel mit $r = \mathtt{GAG}$ und $s = \mathtt{CACG}$:



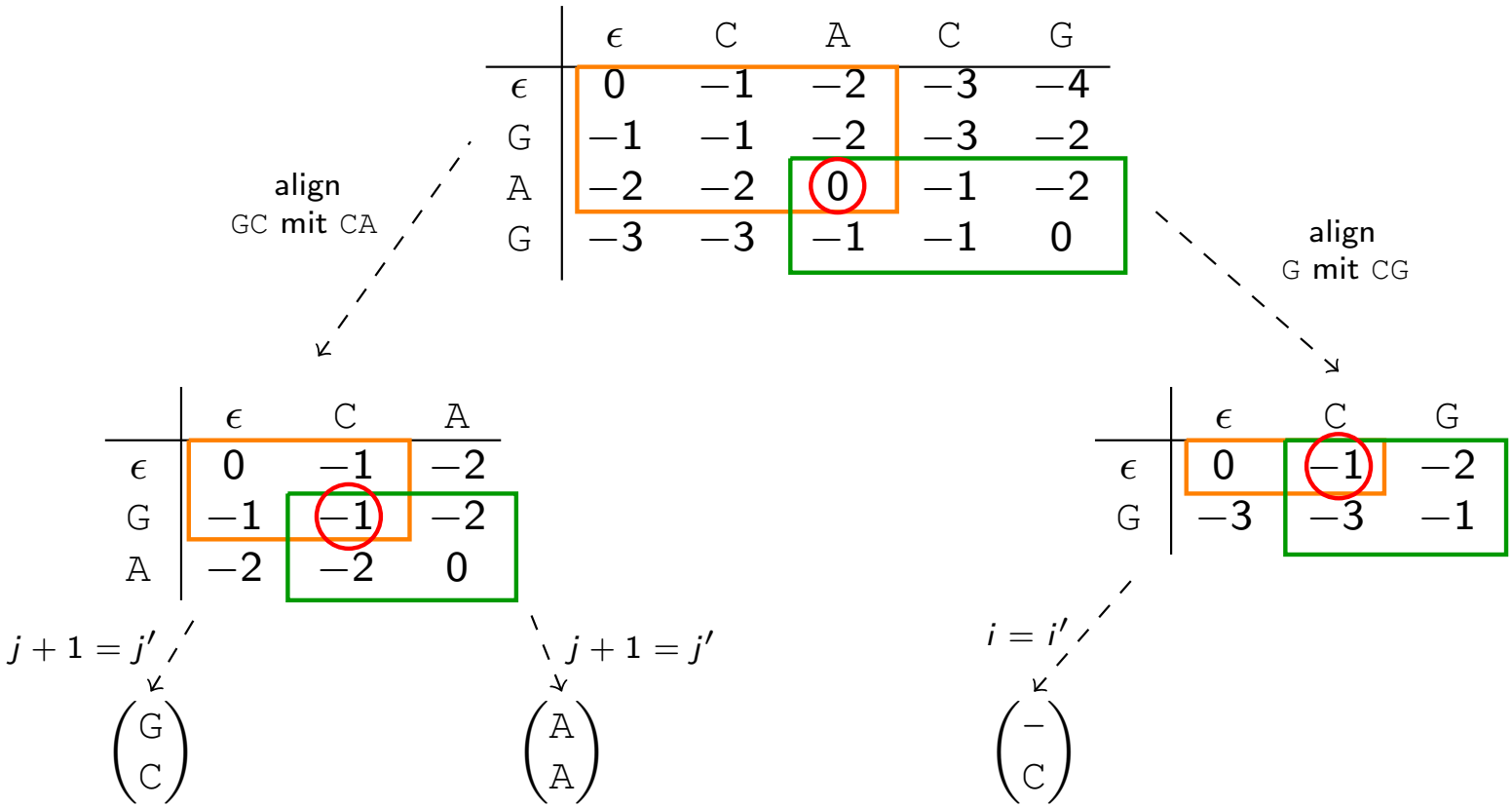
Hirschberg-Algorithmus

Beispiel mit $r = \mathtt{GAG}$ und $s = \mathtt{CACG}$:



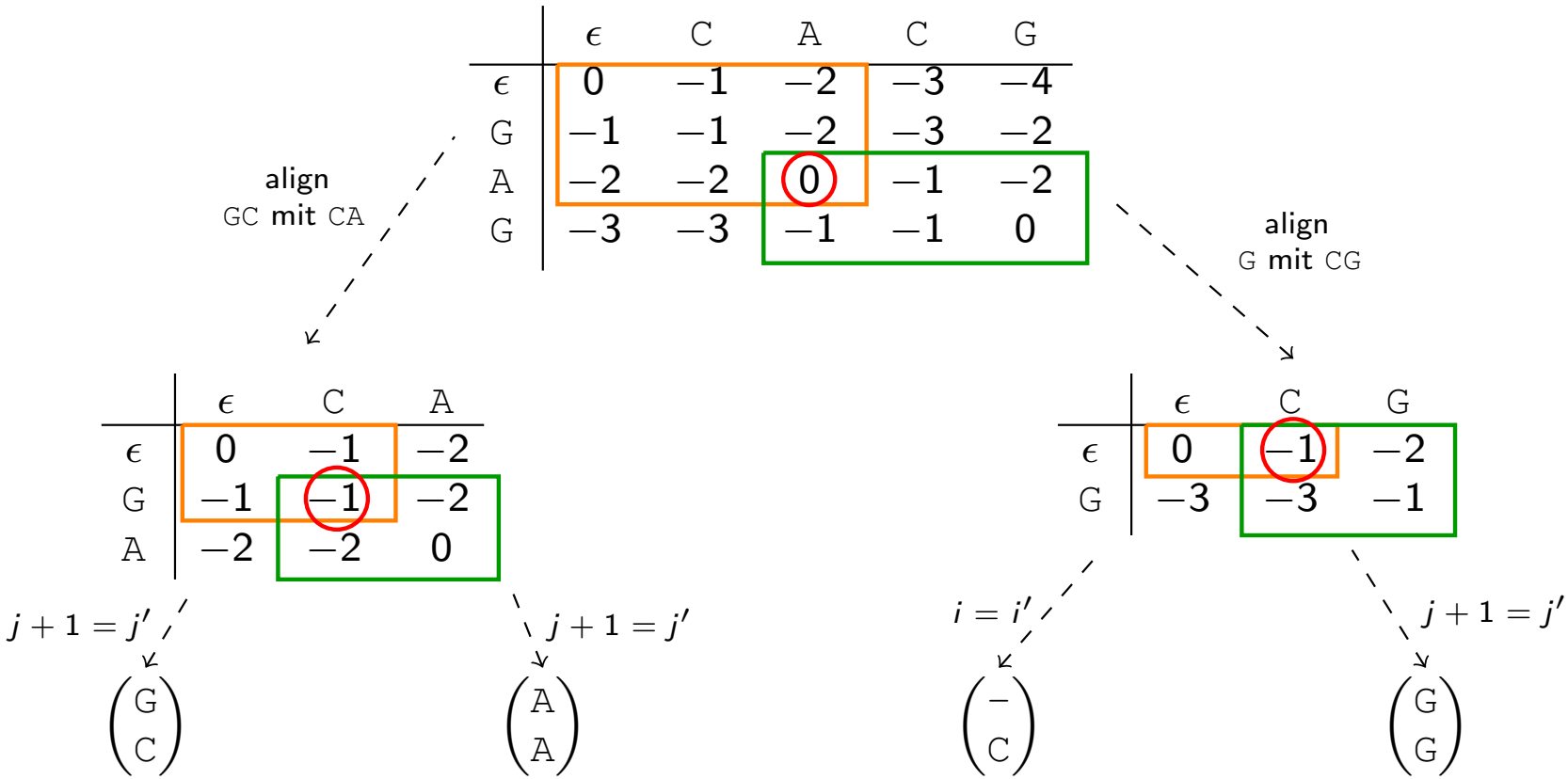
Hirschberg-Algorithmus

Beispiel mit $r = \mathtt{GAG}$ und $s = \mathtt{CACG}$:



Hirschberg-Algorithmus

Beispiel mit $r = \texttt{GAG}$ und $s = \texttt{CACG}$:



Laufzeitanalyse

- In der 0. Iteration werden die Alignments $(0,0) \rightarrow (m, n/2)$ und $(m, n/2) \leftarrow (m, n)$ durchgeführt, die Laufzeit beträgt $O(mn)$.
- Pro Rekursionsschritt wird die Hälfte der DP-Matrix ignoriert.

Laufzeitanalyse

- In der 0. Iteration werden die Alignments $(0, 0) \rightarrow (m, n/2)$ und $(m, n/2) \leftarrow (m, n)$ durchgeführt, die Laufzeit beträgt $O(mn)$.
- Pro Rekursionsschritt wird die Hälfte der DP-Matrix ignoriert.
- In der 1. Iteration wird folglich in der Summe nur $1/2$ Matrix betrachtet, in der 2. Iteration nur $1/4$ Matrix ...

Laufzeitanalyse

- In der 0. Iteration werden die Alignments $(0,0) \rightarrow (m, n/2)$ und $(m, n/2) \leftarrow (m, n)$ durchgeführt, die Laufzeit beträgt $O(mn)$.
- Pro Rekursionsschritt wird die Hälfte der DP-Matrix ignoriert.
- In der 1. Iteration wird folglich in der Summe nur $1/2$ Matrix betrachtet, in der 2. Iteration nur $1/4$ Matrix ...
- Geometrische Reihe: $\sum_{i=0}^{\infty} 2^{-i} mn = 2mn \rightarrow \mathcal{O}(mn)$ Laufzeit.

Implementierung

```
1  def score_col(s, t, sv):
2      ''' returns complete score column '''
3      m, n = len(s), len(t)
4      Sc = [i * sv.Del for i in range(m + 1)]
5      Sl = [0] * (m + 1)
6      for j, tj in zip(range(1, n + 1), t):
7          Sl, Sc = Sc, Sl
8          Sc[0] = j * sv.Ins
9          for i, si in zip(range(1, m + 1), s):
10             Sc[i] = max(Sl[i] + sv.Ins,
11                 Sl[i - 1] + sv.Score(si, tj),
12                 Sc[i - 1] + sv.Del)
13     return Sc
```

Implementierung

```
1  def rev(t): return t[::-1]
2  def hirschberg(s, t, sv, coords = None):
3      if coords == None: coords = (0, 0, len(s), len(t))
4      i1, j1, i2, j2 = coords
5      if i1 == i2 or j1 + 1 == j2:
6          return alignment(s[i1 : i2], t[j1 : j2], sv)
7      jh, cut = (j1 + j2) >> 1, 0 # jh: j-half
8      # score column forward and backward
9      Sf = score_col(s[i1:i2], t[j1:jh], sv)
10     Sb = rev(score_col(rev(s[i1:i2]), rev(t[jh:j2]), sv))
11     # find argmax_i(Sf[i] + Sb[i])
12     for i in range(1, i2 - i1 + 1):
13         if Sf[i] + Sb[i] > Sf[cut] + Sb[cut]: cut = i
14     # recursive step
15     return hirschberg(s, t, sv, (i1, j1, i1 + cut, jh)) \
16          + hirschberg(s, t, sv, (i1 + cut, jh, i2, j2))
```

Zusammenfassung

- Mit dem Hirschberg-Algorithmus ist es möglich ein globales Alignment mit linearem Platzbedarf von $\mathcal{O}(m+n)$ zu berechnen.
- Die asymptotische Laufzeit bleibt weiterhin bei $\mathcal{O}(mn)$.
- Alignment-Varianten wie semiglobales, "Free End Gaps", oder lokales Alignment können diese Methode nutzen, indem bei den Methoden Start- und Endpunkte bestimmt werden und anschießend ein globales Alignment vom Start bis zum Ende durchgeführt wird.

Konzeptionelle Probleme des lokalen Alignments

- Obwohl sich die Nutzung des lokalen Alignments universell durchgesetzt hat, leidet dieses Verfahren an konzeptionellen Problemen.
- Der Grund hierfür besteht in der Definition des lokalen Alignments.
- Die Zielfunktion ist additiv:

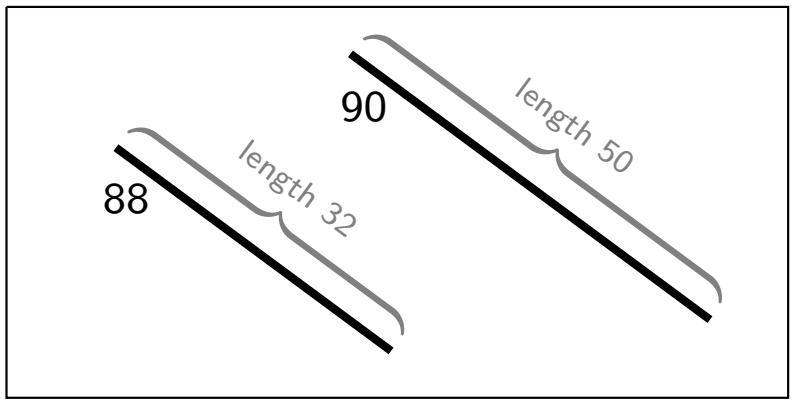
$$score(A) := \sum_{0 \leq i < |A|} score(A_i)$$

Konzeptionelle Probleme des lokalen Alignments

Definition (Schatten-Effekt)

Bei längeren Alignments mit vielen Edit-Operationen können bessere Scores als bei kürzeren exakteren Alignments erzielt werden.

Möglicherweise ist das kürzere Alignment biologisch interessanter.

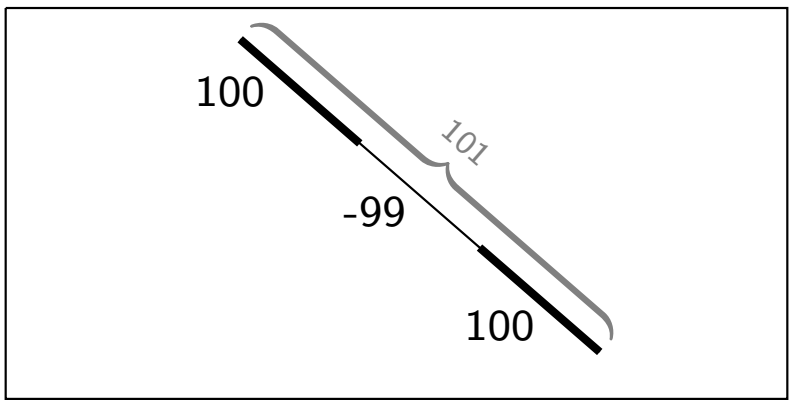


Konzeptionelle Probleme des lokalen Alignments

Definition (Mosaik-Effekt)

Bei sich abwechselnd alignierenden Bereichen (gut, schlecht, gut) kann es dazu kommen, dass der komplette Bereich einen größeren Score hat, als die kürzeren guten Bereiche.

Interessant wären aber eigentlich die guten kurzen Alignments separat.



Konzeptionelle Probleme des lokalen Alignments

- Aufgrund des additiven Verfahrens können lange Alignments mit weniger guten Regionen insgesamt einen besseren Score erhalten als kurze gute Alignments.
- Es liegt nahe einen Score zu suchen, der *längennormalisiert* ist.

Konzeptionelle Probleme des lokalen Alignments

- Aufgrund des additiven Verfahrens können lange Alignments mit weniger guten Regionen insgesamt einen besseren Score erhalten als kurze gute Alignments.
- Es liegt nahe einen Score zu suchen, der *längennormalisiert* ist.
- Achtung: $score(A)/|A|$ wäre für sehr kurze Alignments (etwa eine Spalte / Zeile) kaum zu überbieten.

Konzeptionelle Probleme des lokalen Alignments

Eine pragmatische Lösung ist das Einführen einer Konstanten L, für das Gewichten mit einer Mindestlänge:

$$NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \leq i < |A|} Score(A_i).$$

Konzeptionelle Probleme des lokalen Alignments

Eine pragmatische Lösung ist das Einführen einer Konstanten L, für das Gewichten mit einer Mindestlänge:

$$NormScore_L(A) := \frac{1}{|A| + L} \cdot \sum_{0 \leq i < |A|} Score(A_i).$$

Probleme:

- Um L gut zu bestimmen, müsste schon vor dem Alignieren die Länge der möglichen Alignments bekannt sein.
- Die bekannten DP-Algorithmen können nicht mehr angewendet werden, da der Score nicht mehr additiv ist.

Konzeptionelle Probleme des lokalen Alignments

Arslan und andere (2001) haben einen neuen normalisierten Score vorgestellt:

$$\begin{aligned} DScore_{\lambda,L}(A) &:= Score(A) - \lambda(|A| + L) \\ &= \sum_{i=1}^{|A|} \left(Score(A_i) - \frac{\lambda(|A| + L)}{|A|} \right) \end{aligned}$$

Konzeptionelle Probleme des lokalen Alignments

Arslan und andere (2001) haben einen neuen normalisierten Score vorgestellt:

$$\begin{aligned} DScore_{\lambda,L}(A) &:= Score(A) - \lambda(|A| + L) \\ &= \sum_{i=1}^{|A|} \left(Score(A_i) - \frac{\lambda(|A| + L)}{|A|} \right) \end{aligned}$$

Somit können die herkömmlichen Methoden wieder angewendet werden, wobei der Score nun von λ abhängt.

Konzeptionelle Probleme des lokalen Alignments

- Es kann gezeigt werden, dass es immer ein $\lambda \geq 0$ gibt, so dass die Lösung $DScore_{\lambda,L}(A)$ auch die Lösung von $\max S(A)$ mit $S(A) = score(A)/(|A| + L)$ ist.
- Durch binäre Suche und wiederholtes Alignieren kann ein optimaler Wert für λ gefunden werden, in der Praxis mit 3-5 Iterationen.

Konzeptionelle Probleme des lokalen Alignments

- Es kann gezeigt werden, dass es immer ein $\lambda \geq 0$ gibt, so dass die Lösung $DScore_{\lambda,L}(A)$ auch die Lösung von $\max S(A)$ mit $S(A) = score(A)/(|A| + L)$ ist.
- Durch binäre Suche und wiederholtes Alignieren kann ein optimaler Wert für λ gefunden werden, in der Praxis mit 3-5 Iterationen.
- Trotz der genannten Vorteile hat sich das längennormalisierte Alignment bisher nicht durchgesetzt, vielleicht wegen der Beliebigkeit des Parameters L.

Alignment beschleunigen

- Angenommen, es seien gegeben zwei Sequenzen r, s mit $m := |r|, n := |s|$ und $m = \Theta(n)$ und einem Alphabet Σ mit $\sigma := |\Sigma|$.
- Die herkömmliche Berechnung der Edit-Distanz hätte hierbei eine Laufzeit von $\mathcal{O}(n^2)$.
- Im Folgenden wird eine Technik gezeigt, die diese Berechnung auf subquadratische Laufzeit reduziert.

Alignment beschleunigen

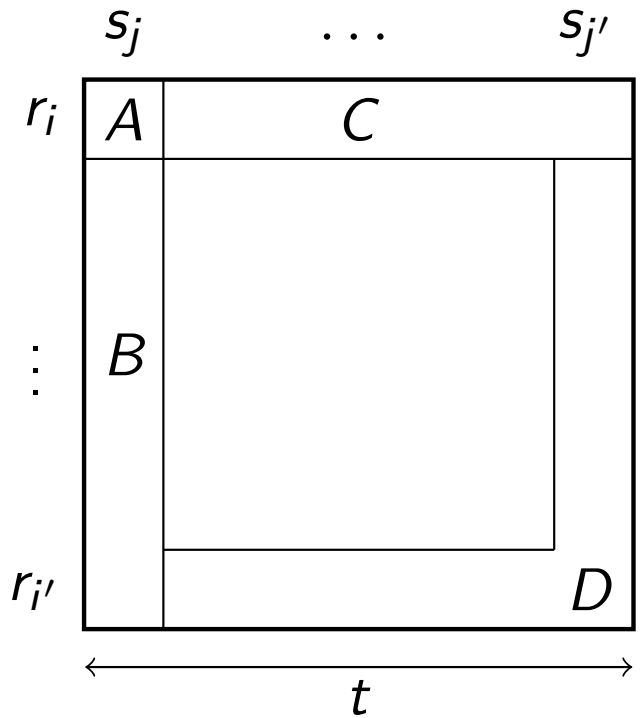
- Angenommen, es seien gegeben zwei Sequenzen r, s mit $m := |r|, n := |s|$ und $m = \Theta(n)$ und einem Alphabet Σ mit $\sigma := |\Sigma|$.
- Die herkömmliche Berechnung der Edit-Distanz hätte hierbei eine Laufzeit von $\mathcal{O}(n^2)$.
- Im Folgenden wird eine Technik gezeigt, die diese Berechnung auf subquadratische Laufzeit reduziert.
- Diese Methode ist auch bekannt als *Method of Four Russians*, benannt nach seinen Entwicklern Arlazarov, Dinic, Kronrod und Faradzev,

Alignment beschleunigen

- Angenommen, es seien gegeben zwei Sequenzen r, s mit $m := |r|, n := |s|$ und $m = \Theta(n)$ und einem Alphabet Σ mit $\sigma := |\Sigma|$.
- Die herkömmliche Berechnung der Edit-Distanz hätte hierbei eine Laufzeit von $\mathcal{O}(n^2)$.
- Im Folgenden wird eine Technik gezeigt, die diese Berechnung auf subquadratische Laufzeit reduziert.
- Diese Methode ist auch bekannt als *Method of Four Russians*, benannt nach seinen Entwicklern Arlazarov, Dinic, Kronrod und Faradzev, (wobei tatsächlich nur einer von ihnen ein Russe war).

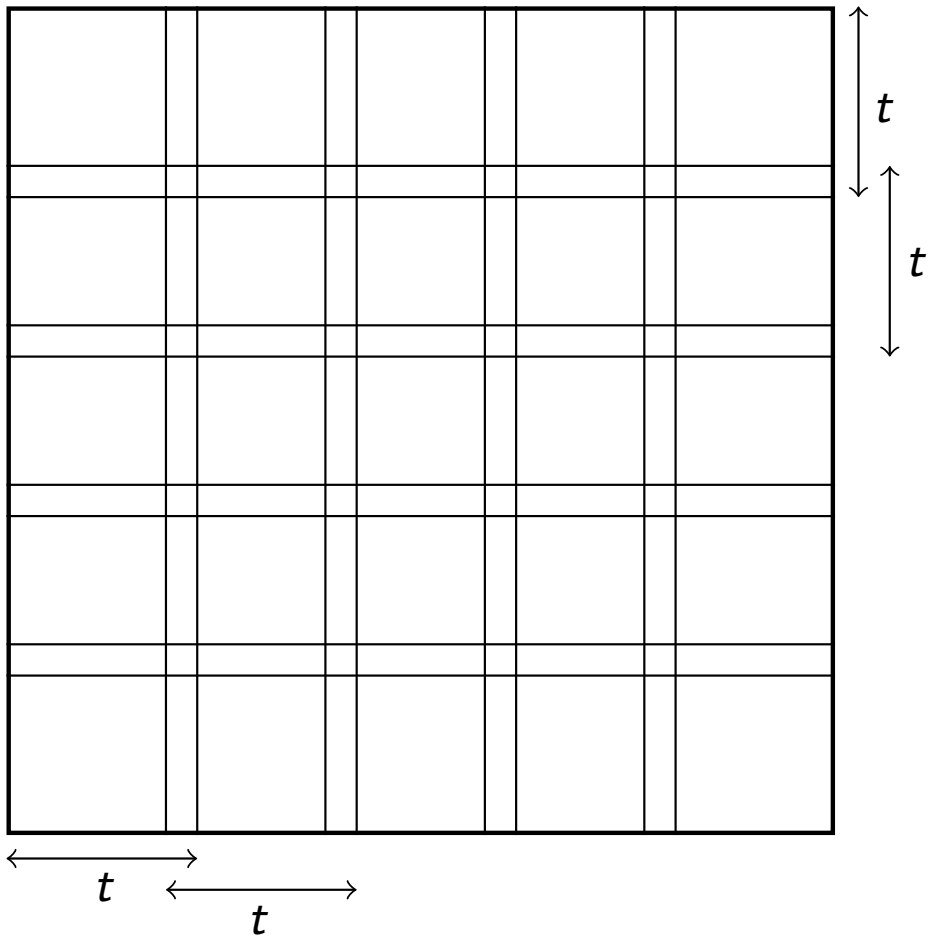
Beobachtung

- Innerhalb eines Rechtecks einer DP-Matrix hängt D nur ab von den Werten A, B, C und den Teilstrings $r[i : i' + 1], s[j : j' + 1]$.
- Definition: Ein t-Block ist ein $t \times t$ großer Ausschnitt einer DP-Matrix.
- Idee: DP-Matrix in t-Blöcke aufteilen und D vorberechnen.



Aufteilung der Matrix

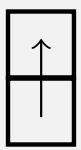
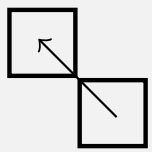
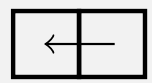
DP-Matrix in überlappende t-Blöcke aufteilen:



Weitere Beobachtung

Lemma

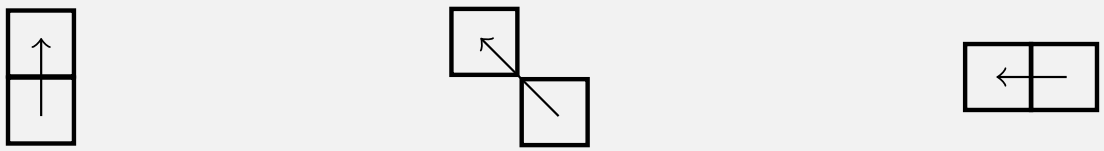
Wenn Einheitskosten gegeben sind, unterscheiden sich zwei adjazente Felder in der DP-Matrix höchstens um Kosten 1.

Weitere Beobachtung

Lemma

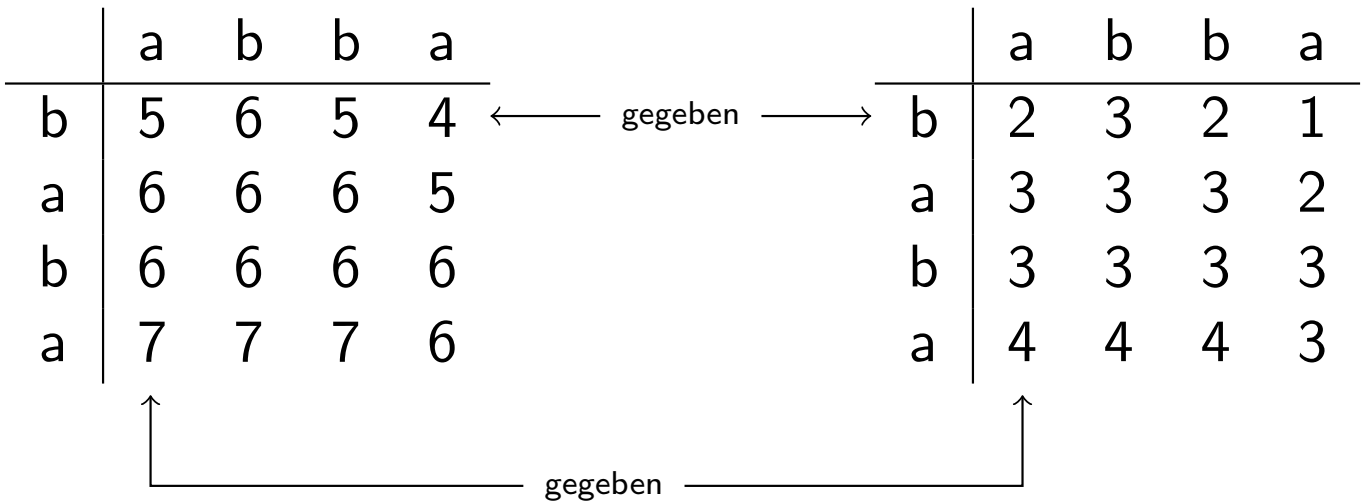
Wenn Einheitskosten gegeben sind, unterscheiden sich zwei adjazente Felder in der DP-Matrix höchstens um Kosten 1.



Der Beweis wurde in vorheriger Vorlesung (Ukkonen's Algorithmus für semiglobales Alignment) erbracht.

Weitere Beobachtung

Es folgt: Wenn A, B, C beim Vergleich zweier t-Blöcke sich jeweils um die selbe Verschiebung v unterscheiden und die Teilstrings identisch sind, dann unterscheiden sich die Werte in D auch um v.



Precomputing

Um zu verhindern, dass man (unendlich) viele Kombinationen von A, B, C Vektoren vorberechnen muss, genügt es einen Offsetvektor V_B, V_C aus B und C zu erzeugen. Sei $V_B[0] := 0$, $V_C[0] := 0$, $V_B[i] := B[i] - B[i-1]$, $V_C[i] := C[i] - C[i-1]$

Precomputing

Um zu verhindern, dass man (unendlich) viele Kombinationen von A, B, C Vektoren vorberechnen muss, genügt es einen Offsetvektor V_B, V_C aus B und C zu erzeugen. Sei $V_B[0] := 0$, $V_C[0] := 0$, $V_B[i] := B[i] - B[i-1]$, $V_C[i] := C[i] - C[i-1]$

	a	b	b	a
b	5	6	5	4
a	6			
b	6			
a	7			

$\rightarrow$

	a	b	b	a
b	0	1	-1	-1
a	1			
b	0			
a	1			

Beispiel:

$$B = 5, 6, 6, 7 \quad \rightarrow \quad V_B = 0, 1, 0, 1$$

$$C = 5, 6, 5, 4 \quad \rightarrow \quad V_C = 0, 1, -1, -1$$

Precomputing

- Weil $V_B[0], V_C[0] = 0$ sind, genügt es $t - 1$ Werte pro Offsetvektor abzuspeichern.
- Da ein Vektoreintrag nur die Werte $\{-1, 0, 1\}$ annehmen kann, kann es $3^{2(t-1)}$ verschiedene Kombinationen geben.

Precomputing

- Weil $V_B[0], V_C[0] = 0$ sind, genügt es $t-1$ Werte pro Offsetvektor abzuspeichern.
- Da ein Vektoreintrag nur die Werte $\{-1, 0, 1\}$ annehmen kann, kann es $3^{2(t-1)}$ verschiedene Kombinationen geben.
- Es gibt insgesamt $\sigma^{2(t-1)}$ verschiedene Substring-kombinationen.
- Die Vorberechnung eines Block-Alignments dauert $(t-1)^2$.

Precomputing

- Weil $V_B[0], V_C[0] = 0$ sind, genügt es $t-1$ Werte pro Offsetvektor abzuspeichern.
- Da ein Vektoreintrag nur die Werte $\{-1, 0, 1\}$ annehmen kann, kann es $3^{2(t-1)}$ verschiedene Kombinationen geben.
- Es gibt insgesamt $\sigma^{2(t-1)}$ verschiedene Substring-kombinationen.
- Die Vorberechnung eines Block-Alignments dauert $(t-1)^2$.
- Insgesamt ergibt sich eine Gesamtlaufzeit für die Vorberechnung von:

$$3^{2(t-1)}\sigma^{2(t-1)}(t-1)^2.$$

Precomputing

- Geschickterweise wählt man $t - 1 = \log_{3\sigma}(n)/2$.

Precomputing

- Geschickterweise wählt man $t - 1 = \log_{3\sigma}(n)/2$.
- Wenn man den Term umformt bekommt man:
  $\mathcal{O}(3^{\log_{3\sigma}(n)} \cdot \sigma^{\log_{3\sigma}(n)} \cdot (\log_{3\sigma}(n))^2)$

Precomputing

- Geschickterweise wählt man $t - 1 = \log_{3\sigma}(n)/2$.
- Wenn man den Term umformt bekommt man:
  $\mathcal{O}(3^{\log_{3\sigma}(n)} \cdot \sigma^{\log_{3\sigma}(n)} \cdot (\log_{3\sigma}(n))^2)$
  $\mathcal{O}((3\sigma)^{\log_{3\sigma}(n)} \cdot (\log_{3\sigma}(n))^2)$

Precomputing

- Geschickterweise wählt man $t - 1 = \log_{3\sigma}(n)/2$.
- Wenn man den Term umformt bekommt man:
  $\mathcal{O}(3^{\log_{3\sigma}(n)} \cdot \sigma^{\log_{3\sigma}(n)} \cdot (\log_{3\sigma}(n))^2)$
  $\mathcal{O}((3\sigma)^{\log_{3\sigma}(n)} \cdot (\log_{3\sigma}(n))^2)$
  $\mathcal{O}\left(n \cdot (\log_{3\sigma}(n))^2\right)$

Precomputing

- Geschickterweise wählt man $t - 1 = \log_{3\sigma}(n)/2$.
- Wenn man den Term umformt bekommt man:
  $\mathcal{O}(3^{\log_{3\sigma}(n)} \cdot \sigma^{\log_{3\sigma}(n)} \cdot (\log_{3\sigma}(n))^2)$
  $\mathcal{O}((3\sigma)^{\log_{3\sigma}(n)} \cdot (\log_{3\sigma}(n))^2)$
  $\mathcal{O}\left(n \cdot (\log_{3\sigma}(n))^2\right) \subset \mathcal{O}(n^2)$

Precomputing

Um D vorzuberechnen, müssen zuerst die Offset-Vektoren wieder umgeformt werden.

	a	b	b	a
b	0	1	-1	-1
a	1			
b	0			
a	1			

$\rightarrow$

	a	b	b	a
b	0	1	0	-1
a	1			
b	1			
a	2			

Precomputing

Um D vorzuberechnen, müssen zuerst die Offset-Vektoren wieder umgeformt werden.

	a	b	b	a
b	0	1	-1	-1
a	1			
b	0			
a	1			

$\rightarrow$

	a	b	b	a
b	0	1	0	-1
a	1			
b	1			
a	2			

Beispiel:

$$V_B = 0,1,0,1 \quad \rightarrow \quad B = 0,1,1,2$$
$$V_C = 0,1,-1,-1 \quad \rightarrow \quad C = 0,1,0,-1$$

Precomputing

Nach der Umformung wird ein herkömmliches globales Alignment angewendet,

$V_B = 0, 1, 0, 1$
$V_C = 0, 1, -1, -1$

		a	b	b	a
	b	0	1	0	-1
	a	1			
	b	1			
	a	2			

Precomputing

Nach der Umformung wird ein herkömmliches globales Alignment angewendet,

$V_B = 0, 1, 0, 1$
$V_C = 0, 1, -1, -1$

	a	b	b	a
b	0	1	0	-1
a	1	1	1	0
b	1	1	1	1
a	2	2	2	1

Precomputing

Nach der Umformung wird ein herkömmliches globales Alignment angewendet,

$V_B = 0, 1, 0, 1$
$V_C = 0, 1, -1, -1$

		a	b	b	a
	b	0	1	0	-1
	a	1	1	1	0
	b	1	1	1	1
	a	2	2	2	1

und in einem Array abgespeichert:

$$1, 0, 1;\quad 1, -1, -1;\quad \texttt{aba};\quad \texttt{bba}\quad \rightarrow\quad 2, 2, 1, 1, 0$$

Precomputing

Nachdem alle D-Vektoren vorberechnet sind, kann das globale Alignment durchgeführt werden. Der Speicherverbrauch beträgt $\mathcal{O}(n \log_{3\sigma}(n))$.

$$
\begin{aligned}
F = \{ & \\
& \vdots \\
1,0,0;\ 1,-1,-1;\ \texttt{aba};\ \texttt{bba} &\ \rightarrow\ 1,2,1,1,0 \\
1,0,1;\ 1,-1,-1;\ \texttt{aba};\ \texttt{bba} &\ \rightarrow\ 2,2,1,1,0 \\
1,1,0;\ 1,-1,-1;\ \texttt{aba};\ \texttt{bba} &\ \rightarrow\ 2,2,1,1,0 \\
& \vdots \\
\} &
\end{aligned}
$$

Globales Alignment

	–	a	b	b	a	b	a
–							
b							
b							
a							
a							
b							
a							

Globales Alignment

1. Initiiere 0-te Zeile und Spalte

$V_B =$
$V_C =$
$A =$
$D =$

		a	b	b	a	b	a
	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Globales Alignment

1. Initiiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:

$V_B =$
$V_C =$
$A = 0$
$D =$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Globales Alignment

1. Initiiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.

$V_B = 0, 1, 1, 1$
$V_C = 0, 1, 1, 1$
$A = 0$
$D =$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Globales Alignment

1. Initiiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup in $F[V_B, V_C, r[i' : i''], s[j' : j'']]$.

$V_B = 0, 1, 1, 1$
$V_C = 0, 1, 1, 1$
$A = 0$
$D = 2, 2, 2, 1, 2$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Globales Alignment

1. Initiiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup in $F[V_B, V_C, r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array.

$V_B = 0, 1, 1, 1$
$V_C = 0, 1, 1, 1$
$A = 0$
$D = 2, 2, 2, 1, 2$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			
b	2			1			
a	3	2	2	2			
a	4						
b	5						
a	6						

Globales Alignment

1. Initiiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup in $F[V_B, V_C, r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array.

$V_B = 0, 1, 1, 1$
$V_C = 0, -1, 0, 0$
$A = 3$
$D = 2, 1, 1, 0, 0$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			
b	2			1			
a	3	2	2	2			
a	4			3			
b	5			3			
a	6	5	4	4			

Globales Alignment

1. Initiiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup in $F[V_B, V_C, r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array.

$V_B = 0, -1, -1, 1$
$V_C = 0, 1, 1, 1$
$A = 3$
$D = -2, -1, 0, 1, 2$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			5
b	2			1			4
a	3	2	2	2	1	2	3
a	4			3			
b	5			3			
a	6	5	4	4			

Globales Alignment

1. Initiiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup in $F[V_B, V_C, r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array.

$V_B = 0, 1, 0, 1$
$V_C = 0, -1, 1, 1$
$A = 2$
$D = 1, 1, 0, 1, 0$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			5
b	2			1			4
a	3	2	2	2	1	2	3
a	4			3			2
b	5			3			3
a	6	5	4	4	3	3	2

Globales Alignment

Laufzeiten:

- Berechnen der Offset-Vektoren: $\mathcal{O}(\log_{3\sigma}(n))$
- Lookup in F: $\mathcal{O}(\log_{3\sigma}(n))$
- Aktualiesierung des D-Vektors: $\mathcal{O}(\log_{3\sigma}(n))$

Globales Alignment

Laufzeiten:

- Berechnen der Offset-Vektoren: $\mathcal{O}(\log_{3\sigma}(n))$
- Lookup in F: $\mathcal{O}(\log_{3\sigma}(n))$
- Aktualiesierung des D-Vektors: $\mathcal{O}(\log_{3\sigma}(n))$
- Laufzeit pro t-Block: $\mathcal{O}(\log_{3\sigma}(n))$
- Anzahl der t-Blöcke: n^2/t^2

Globales Alignment

Laufzeiten:

- Berechnen der Offset-Vektoren: $\mathcal{O}(\log_{3\sigma}(n))$
- Lookup in F: $\mathcal{O}(\log_{3\sigma}(n))$
- Aktualiesierung des D-Vektors: $\mathcal{O}(\log_{3\sigma}(n))$
- Laufzeit pro t-Block: $\mathcal{O}(\log_{3\sigma}(n))$
- Anzahl der t-Blöcke: n^2/t^2

Laufzeit insgesamt: $\mathcal{O}\left(\frac{n^2}{\log_{3\sigma}(n)}\right)$

Zusammenfassung

- Mit dem Four-Russians Trick lässt sich ein globales Alignment in subquadratischer Zeit berechnen.
- Die Laufzeit für das Preprocessing beträgt $O\big(n(\log_{3\sigma}(n))^2\big)$ und für das Alignment $\mathcal{O}(n^2/\log_{3\sigma}(n))$.
- Wegen der hohen Basis im Logarithmus, ist für kleine Alphabete (z.B. $\sigma = 4$ für DNA) die Methode erst ab $n > 10^4$ sinnvoll.
- Daher wird diese Methode in der Praxis nicht genutzt.