Alignment-Verfahren zum Vergleich biologischer Sequenzen

Hans-Joachim Böckenhauer
Dennis Komm

Volkshochschule Zürich

30. April 2014

Ein biologisches Problem

Fragestellung

Finde eine Methode zum Vergleich von DNA-Molekülen oder Proteinen

Motivation

- Suche in Genom- oder Protein-Datenbanken
- Erstellung von phylogenetischen Bäumen
- Teilproblem bei der DNA-Sequenzierung

Vorgehen zur Lösung des Problems

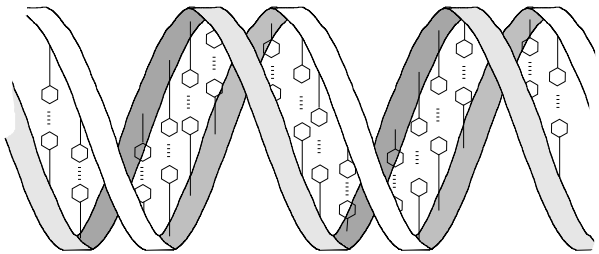
Vorgehen:

- Finde geeignete Datenstruktur für die Moleküle
- Definiere geeignetes Ähnlichkeitsmass
- Entwirf einen möglichst effizienten Algorithmus zur Berechnung der Ähnlichkeit bezüglich dieses Masses

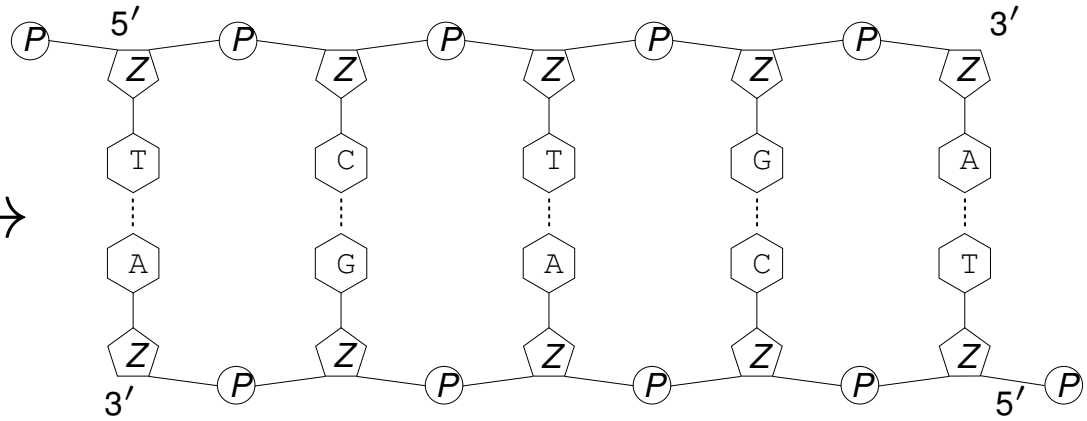
Moleküle als Strings:

DNA und Proteine sind lange kettenförmige Moleküle bestehend aus wenigen oft wiederholten Grundbausteinen

$\Rightarrow$ Darstellung als Strings



$\rightarrow$



$\rightarrow$ TCTGA

Ähnlichkeit von Strings

Anforderung:

Ähnlichkeitsmass soll häufige Veränderungen in DNA- oder Proteinsequenzen widerspiegeln:

- Austausch einzelner Basen oder Aminosäuren
- Einfügen oder Löschen kurzer Teilsequenzen

Alignments

Idee:

Schreibe beide Strings buchstabenweise untereinander, füge dabei an beliebigen Stellen Lückensymbole ein

Beispiel:

Eingabe: Strings $s = \texttt{GACGATTATG}$ und $t = \texttt{GATCGAATAG}$

mögliche Alignments:

$$s' = \texttt{GA-CGATTATG} \qquad s'' = \texttt{GAC-GATTATG}$$
$$t' = \texttt{GATCGAATA-G} \qquad t'' = \texttt{GATCGAATAG-}$$

$$s''' = \texttt{GACGAT----TA-TG}$$
$$t''' = \texttt{----GATCGAATAG--}$$

Bemerkung:

Spalten bestehend aus zwei Lücken sind sinnlos, kommen also nicht vor

Bewertung von Alignments

Idee zur Bewertung:

- Alignment spaltenweise bewerten, dann über alle Spalten aufsummieren
- Spalte mit Lücke erhält Kosten g
- Spalte mit Buchstaben a und b erhält Kosten $p(a, b)$
- $p(a, b)$ ist Null für $a = b$ und gross für $a \neq b$
- Ziel: Minimiere die Kosten

Edit-Distanz

Beispiel für Bewertung:

Edit-Distanz (Levenshtein, 1966): Zähle Mismatches und Lücken, d. h. $g = 1$, $p(a, a) = 0$ und $p(a, b) = 1$ für $a \neq b$

Beispiel:

Eingabe: Strings $s = \texttt{GACGATTATG}$ und $t = \texttt{GATCGAATAG}$

mögliche Alignments:

$$s' = \texttt{GA-CGATTATG} \qquad s'' = \texttt{GAC-GATTATG}$$
$$t' = \texttt{GATCGAATA-G} \qquad t'' = \texttt{GATCGAATAG-}$$

$$s''' = \texttt{GACGAT----TA-TG}$$
$$t''' = \texttt{----GATCGAATAG--}$$

Edit-Distanz:

$d_{\text{edit}}(s', t') = 3 \qquad d_{\text{edit}}(s'', t'') = 5 \qquad d_{\text{edit}}(s''', t''') = 10$

Erster Lösungsversuch: Vollständige Suche

Frage: Wie kann man ein optimales Alignment finden?

Idee: Probiere alle Alignments durch

Abschätzung des Rechenaufwands: Wieviele verschiedene Alignments gibt es?

Anzahl möglicher Alignments

Theorem

Seien s und t zwei Strings der Länge n.
Dann gibt es mehr als 3^n mögliche Alignments von s und t.

Beweisidee:

- Alignment ist eindeutig bestimmt durch die Position der eingefügten Lücken
- Zähle nur Alignments einer bestimmten einfachen Form

Anzahl möglicher Alignments

- Drei Möglichkeiten für zwei Strings a und b der Länge 1:

$$\begin{array}{ccc} a & a- & -a \\ b & -b & b- \end{array}$$

- Sei $s = s_1 s_2 \ldots s_n$ und $t = t_1 t_2 \ldots t_n$
- $\Rightarrow$ Die drei Möglichkeiten anwenden auf alle Paare s_j und t_j
- Beispiel für $n = 3$:

$$\begin{array}{cc|cc|c} s_1 & - & - & s_2 & s_3 \\ - & t_1 & t_2 & - & t_3 \end{array} \quad \text{oder} \quad \begin{array}{c|cc|cc} s_1 & s_2 & - & - & s_3 \\ t_1 & - & t_2 & t_3 & - \end{array}$$

- $\Rightarrow$ Daraus lassen sich 3^n Alignments zusammensetzen
- Beachte: Dies sind nicht alle möglichen Alignments, es gibt noch mehr, zum Beispiel

$$\begin{array}{ccccc} - & - & s_1 & s_2 & s_3 \\ t_1 & t_2 & t_3 & - & - \end{array} \quad \text{oder} \quad \begin{array}{ccccc} s_1 & s_2 & - & - & s_3 \\ - & t_1 & t_2 & t_3 & - \end{array}$$

Exponentielle Laufzeit

n	10	50	100	300	10 000
$10n$	100	500	1 000	3 000	100 000
$4n^2$	400	10 000	40 000	360 000	400 000 000
n^3	1 000	125 000	1 000 000	27 000 000	13 Ziffern
3^n	59 049	24 Ziffern	48 Ziffern	143 Ziffern	4 772 Ziffern

$\Rightarrow$ Vollständige Suche ist viel zu langsam

$\Rightarrow$ Intelligenter Algorithmenentwurf nötig

Dynamische Programmierung

Prinzip der dynamischen Programmierung:
Lösung für die gesamte Eingabe zusammensetzen aus Teillösungen für Teilprobleme, beginnend mit den kleinsten Teilproblemen

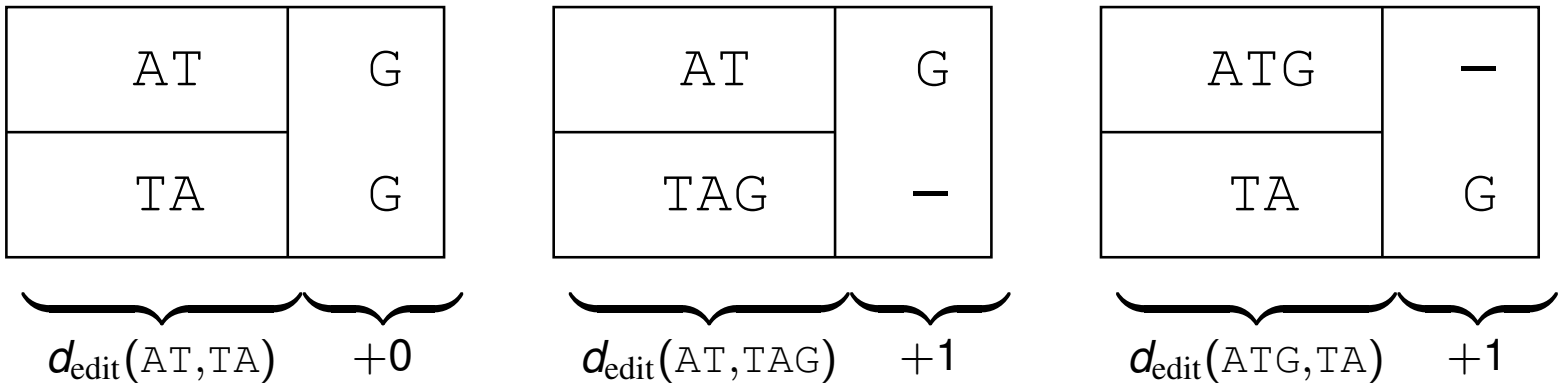
Problem: Finde geeignete Teilprobleme

Idee (Needleman und Wunsch, 1970):

- Alle Paare von Anfangsstücken (Präfixen) der gegebenen Strings als Teilprobleme
- Berechne Alignments für längere Präfixe aus den optimalen Alignments für kürzere Präfixe

Beispiel für das Alignment von Präfixen

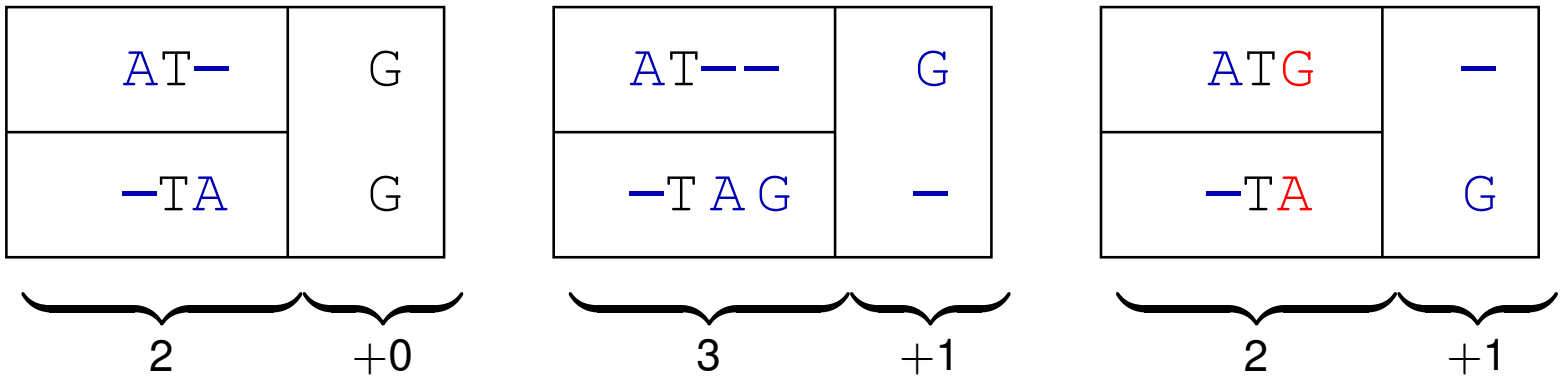
Ziel: Berechne optimales Alignment von $s = \texttt{ATG}$ und $t = \texttt{TAG}$
Unterscheide drei Fälle bezüglich der letzten Spalte des Alignments:



$\Rightarrow$ Berechnung von $d_{\text{edit}}(\texttt{ATG}, \texttt{TAG})$ zurückgeführt auf Berechnung der Edit-Distanz für drei Paare von Präfixen

Beispiel für das Alignment von Präfixen

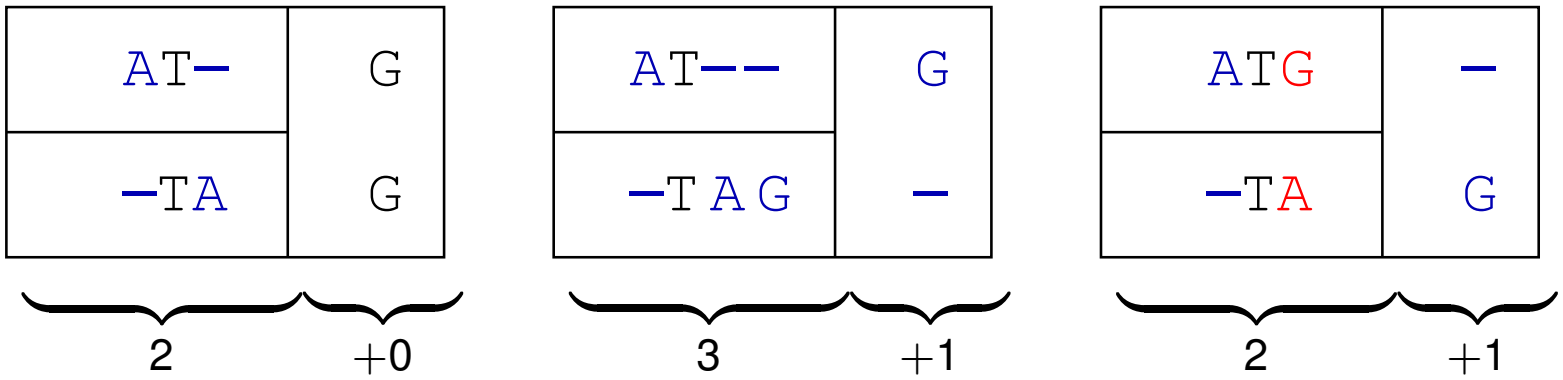
Ziel: Berechne optimales Alignment von $s = \texttt{ATG}$ und $t = \texttt{TAG}$
Unterscheide drei Fälle bezüglich der letzten Spalte des Alignments:



$\Rightarrow$ Berechnung von $d_{\text{edit}}(\texttt{ATG}, \texttt{TAG})$ zurückgeführt auf Berechnung der Edit-Distanz für drei Paare von Präfixen

Beispiel für das Alignment von Präfixen

Ziel: Berechne optimales Alignment von $s = \texttt{ATG}$ und $t = \texttt{TAG}$
Unterscheide drei Fälle bezüglich der letzten Spalte des Alignments:



$\Rightarrow$ optimales Alignment mit Edit-Distanz $d_{\text{edit}}(s', t') = 2$ ist

$$\begin{aligned} s' &= \texttt{AT−G} \\ t' &= \texttt{−TAG} \end{aligned}$$

Initialisierung

Definition: Der leere String λ ist ein String der Länge 0, er ist Präfix von jedem anderen String.

Initialisierung der Berechnung:

Alignment eines nichtleeren Präfixes mit dem leeren String ist eindeutig:

$$\begin{array}{c} s_1\ s_2\ \ldots\ s_i \\ -\ -\ \ldots\ - \end{array} \quad \text{oder} \quad \begin{array}{c} -\ -\ \ldots\ - \\ t_1\ t_2\ \ldots\ t_i \end{array}$$

Kosten: $d_{\text{edit}}(s_1 \ldots s_i, \lambda) = d_{\text{edit}}(\lambda, t_1 \ldots t_i) = i$

Beispiel zur Berechnung der Edit-Distanz

S \ t		C	C	T	G
	0	1	2	3	4
	0				
A	1	1			
C	2	2			
T	3	3			
T	4	4			
G	5	5			

Initialisierung

Beispiel zur Berechnung der Edit-Distanz

S \ t		C	C	T	G
	0	1	2	3	4
A	1	1			
C	2				
T	3				
T	4				
G	5				

+1 +1 +1

Bestimme $d_{\text{edit}}(s_1, t_1)$:

Lücke in t einfügen

$$\underbrace{\begin{matrix}-\\ \mathtt{C}\end{matrix}}_{d_{\text{edit}}(\lambda, t_1)} \Bigg| \underbrace{\begin{matrix}\mathtt{A}\\ -\end{matrix}}_{+1}$$

Lücke in s einfügen

$$\underbrace{\begin{matrix}\mathtt{A}\\ -\end{matrix}}_{d_{\text{edit}}(s_1, \lambda)} \Bigg| \underbrace{\begin{matrix}-\\ \mathtt{C}\end{matrix}}_{+1}$$

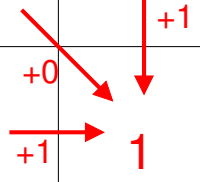
Mismatch einfügen

$$\underbrace{\begin{matrix}\lambda\\ \lambda\end{matrix}}_{d_{\text{edit}}(\lambda, \lambda)} \Bigg| \underbrace{\begin{matrix}\mathtt{A}\\ \mathtt{C}\end{matrix}}_{+1}$$

Minimum bilden

Beispiel zur Berechnung der Edit-Distanz

S \ t		C	C	T	G
	0	1	2	3	4
A 1	1	1			
C 2	2	1			
T 3	3				
T 4	4				
G 5	5				



Berechne $d_{\text{edit}}(s_2, t_1)$

Beispiel zur Berechnung der Edit-Distanz

S \ t		C	C	T	G
	0	1	2	3	4
A	1	1			
C	2	1			
T	3	2			
T	4	3			
G	5	4			

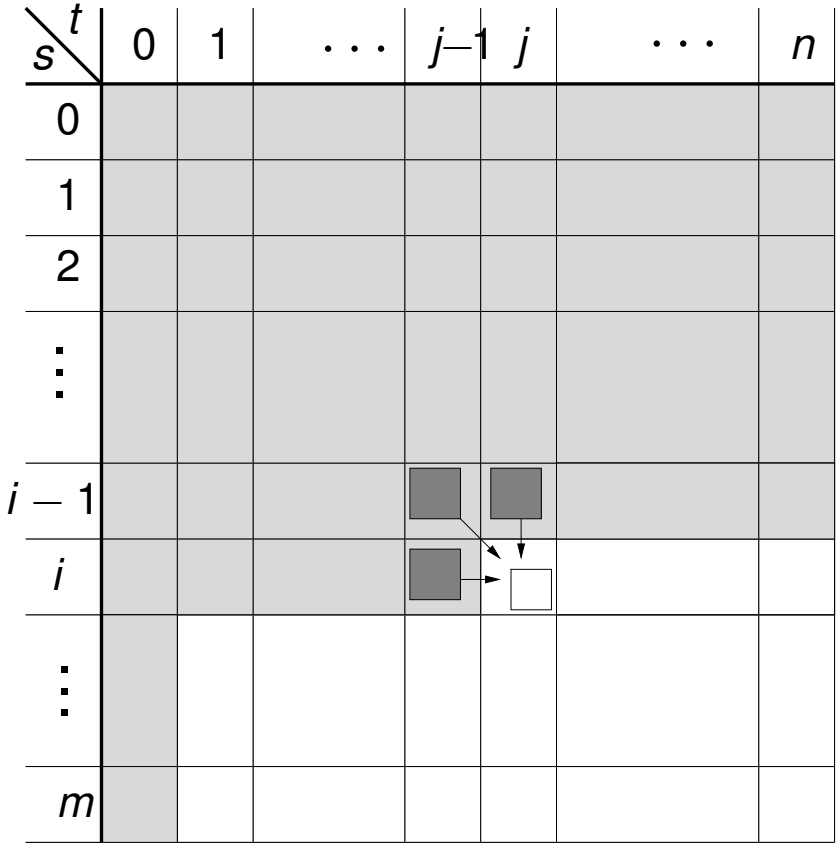
Berechne den Rest von Spalte 1

Beispiel zur Berechnung der Edit-Distanz

S \ t		C	C	T	G
	0	1	2	3	4
A	1	1	2	3	4
C	2	1	1	2	3
T	3	2	2	1	2
T	4	3	3	2	2
G	5	4	4	3	2

$$d_{\text{edit}}(s, t) = 2$$

Darstellung als Matrix



Letzte Spalte des Alignments ist

↓ Lücke in t

→ Lücke in s

↘ Match/Mismatch

$$d_{\text{edit}}(s_1 \dots s_i, t_1 \dots t_j) = \min\{d_{\text{edit}}(s_1 \dots s_{i-1}, t_1 \dots t_j) + 1,$$
$$d_{\text{edit}}(s_1 \dots s_i, t_1 \dots t_{j-1}) + 1,$$
$$d_{\text{edit}}(s_1 \dots s_{i-1}, t_1 \dots t_{j-1}) + p(s_i, t_j)\}$$

Analyse der Laufzeit

Eingabe: Zwei Strings $s = s_1 \dots s_m$ und $t = t_1 \dots t_n$

Initialisierung der Ränder: $m + n + 2$ Operationen

for $i = 0$ **to** m **do**
  $M(i, 0) = i$
for $j = 0$ **to** n **do**
  $M(0, j) = j$

Füllen der Matrix: $4 \cdot m \cdot n$ Operationen

for $i = 1$ **to** m **do**
  for $j = 1$ **to** n **do**

$$M(i,j) := \min\{M(i-1,j)+1,\; M(i,j-1)+1,\; M(i-1,j-1)+p(s_i,t_j)\}$$

Ausgabe: $d_{\text{edit}}(s, t) = M(m, n)$

Analyse der Laufzeit

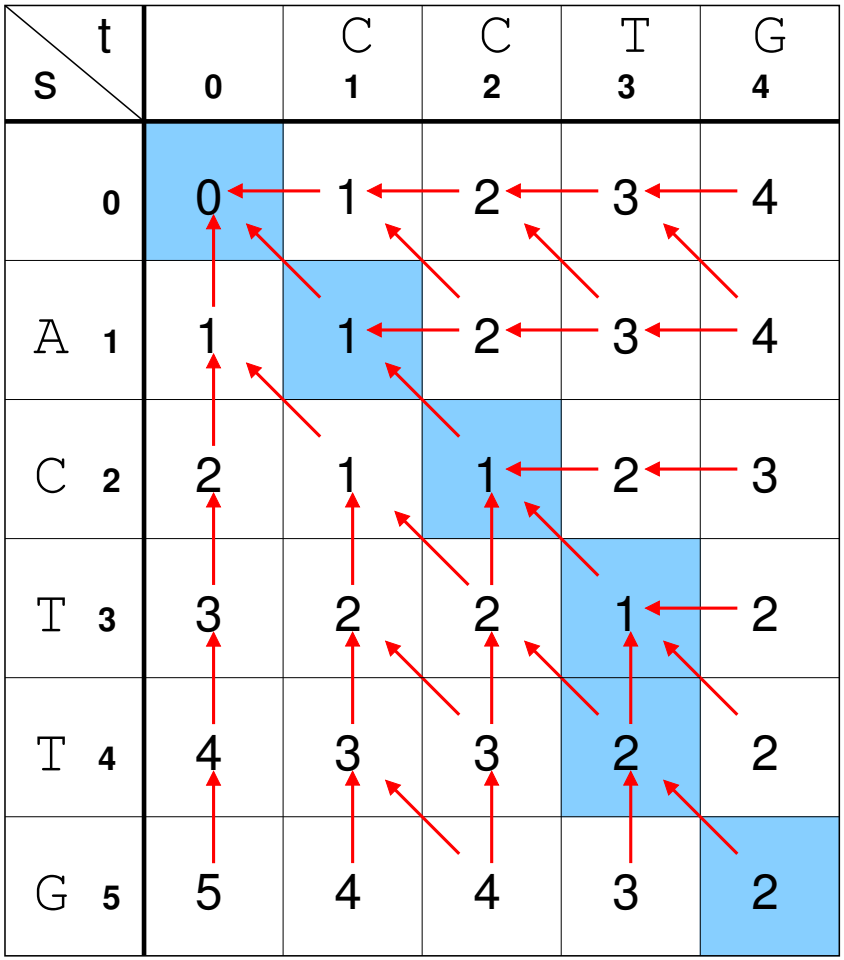
Laufzeit: ungefähr $4n^2$ für zwei Strings der Länge n

n	10	50	100	300	10 000
$10n$	100	500	1 000	3 000	100 000
$4n^2$	400	10 000	40 000	360 000	400 000 000
n^3	1 000	125 000	1 000 000	27 000 000	13 Ziffern
3^n	59 049	24 Ziffern	48 Ziffern	143 Ziffern	4 772 Ziffern

Anschaulich: Vergleich zweier Gene (Länge in der Grössenordnung von 10 000) braucht

- 100 MB Platz und
- $<$ 1 Minute Zeit

Bestimmung des optimalen Alignments



$$s' = \mathtt{A\,C\,T\,T\,G}$$

$$t' = \mathtt{C\,C\,T\,-\,G}$$

Problem bei Strings ungleicher Länge

Beispiel: Betrachte die Strings $s = \texttt{TAAGGT}$ und $t = \texttt{AGTTTATAGCCTGGT}$

Ein optimales Alignment mit Edit-Distanz $d_{\text{edit}}(s', t') = 9$ ist

$$\begin{aligned} s' &= \texttt{---TA-A---GGT} \\ t' &= \texttt{AGTTTATAGCCTGGT} \end{aligned}$$

Kompakter und besser biologisch motiviert ist

$$\begin{aligned} s'' &= \texttt{---TA-AGG-T---} \\ t'' &= \texttt{AGTTTATAGCCTGGT} \end{aligned}$$

mit Edit-Distanz $d_{\text{edit}}(s'', t'') = 10$

Strings ungleicher Länge vergleichen

Idee: Lücken am Beginn und Ende des kürzeren Strings nicht mitzählen

Umsetzung (für $|s| < |t|$):

- erste Zeile der Matrix mit Nullen initialisieren
  $\Rightarrow$ Lücken vor dem Lesen von s_1 kostenlos
- Minimum der Werte in der letzten Zeile liefert das Ergebnis
  $\Rightarrow$ Lücken nach dem Lesen von s_m kostenlos
- Rest des Algorithmus wie vorher

Beispiel

S \ t		C 1	C 2	A 3	C 4	T 5	T 6	T 7	G 8	T 9
	0	0	0	0	0	0	0	0	0	0
C 1	1	0	0	1	0	1	1	1	1	1
A 2	2	1	1	0	1	1	2	2	2	2
G 3	3	2	2	1	1	2	2	3	2	3
T 4	4	3	3	2	2	1	2	2	3	2

$$s' = -\mathtt{CAGT}----$$

$$t' = \mathtt{CCACTTTGT}$$

Ausblick: Andere Erweiterungen

- Lokales Alignment (Smith und Waterman, 1981): Finde Teilstrings mit maximaler Ähnlichkeit, ignoriere dabei nicht passende Anfangs-und Endstücke
- Lücken stärker gewichten als Mismatches (kommen auch in der Natur seltener vor)
- Verschiedene Gewichtung von Mismatches, zum Beispiel entsprechend der chemischen Ähnlichkeit der Aminosäuren beim Vergleich von Proteinsequenzen
- Zusätzliche Kosten für das Öffnen einer Sequenz von Lücken

$\Rightarrow$ leichte Abwandlungen des Algorithmus lösen auch diese Probleme

Ausblick: Heuristiken

Problem: Quadratische Laufzeit ist für sehr lange Strings (zum Beispiel ganze Genome) zu gross

Ausweg: Finde Algorithmen mit linearer Laufzeit, die ein gutes (aber nicht notwendigerweise optimales) Alignment finden

Ideen:

- Beschränke die Anzahl der vorkommenden Lückensymbole

  $\Rightarrow$ Berechne nur Streifen konstanter Breite der Matrix um die Mitteldiagonale herum
- Finde zunächst kurze gemeinsame Teilstrings, erweitere diese mit dynamischer Programmierung an beiden Enden, setze diese Teil-Alignments zusammen

Zusammenfassung

- Effizienter Algorithmus für den Vergleich von DNA-Sequenzen
- Modellierung biologischer Fragestellungen als Informatik-Problem
- Algorithmische Technik der dynamischen Programmierung
- Laufzeitanalyse von Algorithmen