

# Der Needleman-Wunsch-Algorithmus

Praktikum Algorithmen in der Bioinformatik

Rolf Haynberg - 65109948

## 1 Motivation und Hintergrund

Im Hintergrund des Needleman-Wunsch-Algorithmus steht das Problem, die Editier-Distanz zweier gegebener Zeichenketten zu finden. Vorkenntnisse zur Editier-Distanz (Levenshtein-Distanz) werden daher im Folgenden empfohlen.

Das Finden einer Editier-Vorschrift und damit auch der Editier-Distanz spielt unter anderem für die Bioinformatik eine wichtige Rolle weil es das Problem des Sequenz-Alignments beschreibt. Lässt sich eine hohe Ähnlichkeit zwischen biomolekularen Sequenzen nachweisen, so impliziert dies gewöhnlich auch eine funktionale oder strukturelle Ähnlichkeit [Gus97].

## 2 Der Algorithmus

Als Eingabe erhält der Algorithmus zwei Zeichenketten  $S_1$  und  $S_2$  mit Länge  $n$  bzw.  $m$ , zwischen denen die Editier-Vorschrift gefunden werden soll. Dabei sei  $S[i]$  das  $i$ -te Zeichen und  $S[1..i]$  das Präfix der Länge  $i$ . Wir definieren außerdem Hilfsweise,  $S[1..0]$  als leeres Wort und  $S[0]$  als ein spezielles Zeichen  $\perp$ , welches nicht im verwendeten Alphabet enthalten ist.

Des Weiteren greift der Algorithmus auf Kostenfunktionen zu, welche die Kosten (in der Literatur wird häufig auch der Begriff 'Gewichte' verwendet) für jede mögliche Editier-Operation definieren (genauer wird weiter unten beschrieben).

Die Ausgabe des Algorithmus ist die Editier-Distanz zwischen  $S_1$  und  $S_2$  sowie eine Editier-Vorschrift. Eine Editier-Vorschrift ist eine Folge von Editier-Operationen mit den geringsten Gesamtkosten. Die Gesamtkosten einer Editier-Vorschrift sind genau die Editier-Distanz. Es ist durchaus möglich, dass es zu einer Eingabe mehrere Editier-Vorschriften gibt.

### 2.1 Funktionsweise

Der Ablauf des Algorithmus besteht im Grunde darin, dynamisch eine  $(n+1) \times (m+1)$  Matrix  $D$  zu erstellen, sodass  $D_{i,j}$  gleich der Editier-Distanz zwischen  $S_1[1..i]$  und  $S_2[1..j]$  ist. Die Editier-Distanz zwischen  $S_1$  und  $S_2$  ist demnach  $D_{n,m}$ .

Beim Erstellen der Matrix geht der Algorithmus iterativ vor (z.B. zeilenweise), beginnend bei  $D_{0,0}$ . In jedem Schritt wird der einzutragende Wert berechnet als

$$D_{0,0} = 0 \tag{1}$$

$$D_{i,j} = \min \begin{cases} D_{i,j-1} + \text{insert}(S_2[j]) & j \geq 1 & \text{(2a)} \\ D_{i-1,j} + \text{delete}(S_1[i]) & i \geq 1 & \text{(2b)} \\ D_{i-1,j-1} + \text{replacematch}(S_1[i], S_2[j]) & i, j \geq 1 & \text{(2c)} \end{cases}$$

Dabei sind *insert*, *delete* und *replacematch* die bereits erwähnten Kostenfunktionen. Sie geben die Kosten für Einfüge-, Lösch-, Ersetzungs- und Match-Operationen zu jedem Zeichen bzw. zu jeder Zeichenkombination zurück. Diese Funktionen wurden hier zur Übersichtlichkeit eingeführt und werden in der Praxis gewöhnlich zusammengefasst und als Tabelle implementiert.

Aus den Fällen in Gleichung (2) wird das Minimum ausgewählt um die geringsten Kosten für jede Zelle zu erhalten. Obwohl nur drei Werte verglichen werden, enthält die Zelle schließlich die geringsten Kosten *aller möglichen* Editier-Vorschriften. Dies lässt sich mit dem Optimalitätsprinzip von Bellman [Wik09b] erklären: Die Berechnung der Werte bauen nämlich auf bereits optimale Teilergebnisse auf. Der Needleman-Wunsch-Algorithmus nutzt also das Paradigma der dynamischen Programmierung [Wik09a].

Der Grundzustand  $D_{0,0} = 0$  behandelt den Fall in dem noch keine Operation ausgeführt wurde. Die Definition ist in sofern sinnvoll, als dass sie die Editier-Distanz zu Beginn des Algorithmus angibt welche unabhängig von der Eingabe immer 0 sein soll.

Des weiteren enthält die obige Berechnungsvorschrift Bedingungen. Ohne diese Bedingungen würden sich bei Berechnungen mit  $i = 0$  (erste Zeile) oder  $j = 0$  (erste Spalte) negative Indizes für  $D$  ergeben, z.B. in (2c). Dies soll vermieden werden da  $D$  für negative Indizes nicht definiert ist. Bei genauerer Betrachtung trifft für die Werte der ersten Zeile ( $i = 0$ ) nur Fall (2a) zu und für die Werte der ersten Spalte ( $j = 0$ ) nur Fall (2a). Alternativ könnte  $D_{x,y}$  für negative  $x$  oder  $y$  als unendlich angenommen werden. Dadurch würde der Fall bei der Minimumsbildung ebenfalls keine Berücksichtigung finden (unendlich würde nie das Minimum sein), was zum selben Ergebnis führt.

## 2.2 Das Traceback

Der Algorithmus kann bisher zwar die Editier-Distanz zwischen den zwei Zeichenketten  $S_1$  und  $S_2$  berechnen, Ziel war es jedoch eine Editier-Vorschrift auszugeben. Diese lässt sich mit relativ wenig Aufwand rekonstruieren wenn während der Ausführung für jede Zelle gespeichert wird, durch welche Operation die Editier-Distanz, also der Wert in der Zelle, zustande kam. Der Algorithmus speichert deshalb für jede Zelle, welcher der vier Fälle (insert, deletion, match, replace) das Minimum bilden. Wird in mehreren Fällen das Minimum erreicht, so werden alle zutreffenden Fälle gespeichert. Mit diesen Informationen lässt sich später die Entwicklung des Wertes  $D_{n,m}$  zurückverfolgen und die Editier-Vorschrift rekonstruieren.

Das Rekonstruieren wird Traceback genannt. Der Traceback-Algorithmus beginnt an der Zelle  $D_{n,m}$  und terminiert beim Erreichen der Zelle  $D_{0,0}$ . Dabei geht er schrittweise vor. Welche Zelle von einer gegebenen Zelle erreichbar sind, wird durch die in der Zelle gespeicherten Informationen festgelegt. Führte z.B. eine Insert-Operation von Wert zum Wert  $D_{i,j}$ , so ist ein Schritt von Zelle  $(i, j)$  zu Zelle  $(i, j - 1)$  möglich. Dies ergibt sich aus den Berechnungsvorschriften aus Gleichung (2).

Für jeden Schritt speichert der Traceback-Algorithmus die dazugehörige Operation und so entsteht nach und nach die Editier-Vorschrift von  $S_1$  zu  $S_2$  in umgekehrter Reihenfolge. Wenn ein Wert gleichermaßen durch verschiedene Operationen erzeugt wird, gibt es auch mehrere mögliche Schritte die von der entsprechenden Zelle ausgehen. Aus diesem Grund gibt es häufig mehrere *Rückwege* von denen alle gültige, aber unterschiedliche Editier-Vorschriften repräsentieren.

## 2.3 Maximierungsvariante

Das Ziel des Algorithmus ist es, die niedrigsten Kosten zu erzielen, denn die geringsten Gesamtkosten sind gerade die Editier-Distanz. Diese kann als ein Maß für die Unter-

schiedlichkeit zweier Zeichenketten verstanden werden. In der Praxis wird hingegen häufig die Ähnlichkeit zweier Zeichenketten gesucht. Diese Diskrepanz zwischen Anforderung und Ergebnis ist jedoch nicht tiefgreifend, weil eine geringe Unterschiedlichkeit eine hohe Ähnlichkeit bedeutet und umgekehrt.

Es überrascht daher nicht, dass der Algorithmus bereits mit unkomplizierten Änderungen ein Maß der Ähnlichkeit ausgibt. Eine dieser Änderungen ist, statt Kostenfunktionen so genannte *Similarity-Funktionen* zu verwenden. Die Similarity-Funktionen sind ebenfalls für die vier möglichen Operationen Ersetzen, Löschen, Match und Einfügen definiert. Der Unterschied zu den Kostenfunktionen ist, dass sie gewöhnlich einen relativ hohen Wert für Operationen zurückgeben die relativ geringe Kosten erzeugen. Eine einfache Möglichkeit um aus gegebenen Kostenfunktionen Similarity-Funktionen zu erhalten wäre, die Ausgabe-werte zu negieren. Das Ziel des Algorithmus ist dementsprechend, eine möglichst hohe *Similarity-Score* zu erzielen. In der Berechnungsvorschrift (2) wird deswegen bei dieser Variante des Algorithmus das *Maximum* der drei Werte gebildet. Die Anfangsbedingung (1) bleibt gleich.

### 3 Beispiel

Betrachten wir nun ein konkretes Beispiel mit

$$S_1 = \text{W U R Z E L}$$

$$S_2 = \text{V I E R T E L}$$

Die Kostenfunktionen seien

$$\text{insert}(x) \equiv \text{delete}(x) \equiv 1$$

und

$$\text{replacematch}(x, y) = \begin{cases} 0 & \text{falls } x = y \\ 1 & \text{sonst} \end{cases}$$

Aus der Definition (Gleichung 1) können wir bereits  $D_{0,0} = 0$  ablesen. Für die Berechnung der Werte in Zeile 0 trifft immer nur Fall 1 (Gleichung 2a) zu, da in den anderen Fällen die Bedingungen nicht erfüllt sind (denn in der ersten Zeile ist  $i = 0$ ). Analog dazu trifft für die Berechnung der Werte in Spalte 0 immer nur Fall 2 (Gleichung 2b) zu da dort  $j = 0$  ist. Damit ergibt sich bereits die Matrix in Abbildung 1.

	⊥	W	U	R	Z	E	L
⊥	0	1	2	3	4	5	6
V	1						
I	2						
E	3						
R	4						
T	5						
E	6						
L	7						

Abbildung 1: Die Editier-Matrix  $D$  zu den Zeichenketten  $S_1 = \text{WURZEL}$  und  $S_2 = \text{VIERTTEL}$ . Die erste Zeile sowie erste Spalte sind bereits ausgefüllt. Die Kostenfunktionen sind in diesem Beispiel unabhängig vom zu löschenden bzw. einzufügenden Zeichen. Zur Übersichtlichkeit sind die Zeichenketten an den Rändern nochmals aufgetragen

Jetzt wird die Matrix nach und nach ausgefüllt. Dabei werden immer Zellen  $(i, j)$  berechnet, bei denen  $(i - 1, j - 1)$ ,  $(i - 1, j)$  und  $(i, j - 1)$  bereits bekannt sind. Solche Zellen lassen sich immer finden (dies ist z.B. der Fall wenn zeilenweise vorgegangen wird).  $D_{1,1}$  berechnen sich zum Beispiel als  $D_{1,1} = \min\{1 + 1, 1 + 1, 0 + 1\} = 1$ . Setzt man dies fort, erhält man schließlich die Matrix aus Abbildung 2. Aus der Abbildung lässt sich die

Editier-Distanz 4 ablesen sowie drei verschiedene Editier-Vorschriften. Eine davon ist zum Beispiel: `replace, replace, insert, match, replace, match, match`

	⊥	W	U	R	Z	E	L
⊥	0	1	2	3	4	5	6
V	1	1	2	3	4	5	6
I	2	2	2	3	4	5	6
E	3	3	3	3	4	4	5
R	4	4	4	3	4	5	5
T	5	5	5	4	4	5	6
E	6	6	6	5	5	4	5
L	7	7	7	6	6	5	4

Abbildung 2: Die vollständig ausgefüllte Editier-Matrix  $D$  für  $S_1$  und  $S_2$ . In Zelle  $(7, 6)$  ist die Editier-Distanz 4 abzulesen. Die Pfeile geben zu treffende Fälle für die Berechnung des Minimums an. Ein waagerechter Pfeil bedeutet eine Einfüge-Operation, ein senkrechter Pfeil eine Löschen-Operation und ein diagonaler Pfeil kann sowohl eine Match- als auch eine Ersetzungs-Operation bedeuten (welche Operation verwendet wurde, wird aus dem Kontext klar). Die Pfeile für gültige Editier-Vorschriften sind rot gefärbt.

## 4 Zusammenfassung

Der Needleman-Wunsch-Algorithmus enthält einen grundlegenden, vielseitig erweiterbaren Ansatz um die Editier-Vorschrift zwischen zwei Zeichenketten zu berechnen. Durch Wahl der Kostenfunktionen kann der Algorithmus bereits an verschiedene Anwendungen angepasst werden. Für die Bioinformatik sind jedoch noch weitere Aspekte wie z.B. Gap-Bildung oder das Konzept der lokalen Alignments, interessant.

Viele, auf das Sequenz-Alignement spezialisierte Algorithmen beruhen auf der Idee des Needleman-Wunsch-Algorithmus (z.B. der Smith-Waterman-Algorithmus [Hay09b] oder der Gotoh-Algorithmus [Hay09a]). Er ist deshalb, trotz seiner Betagtheit, auch heute noch von zentraler Bedeutung auf diesem Gebiet.

### Laufzeitanalyse

Um den Wert einer Zelle  $D_{i,j}$  zu berechnen müssen die drei Nachbarzellen  $D_{i-1,j}$ ,  $D_{i,j-1}$ ,  $D_{i-1,j-1}$  untersucht werden sowie die Strings an den Positionen  $i - 1$  und  $j - 1$ . Diese Kosten können als Konstant angenommen werden (z.B. im RAM-Modell). Die Matrix enthält  $(n + 1) \times (m + 1)$  Zellen, sodass sich insgesamt eine Laufzeit in  $O(nm)$  ergibt.

### Literatur

- [Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*, chapter 11, pages 215–253. Cambridge University Press, first edition edition, 1997.
- [Hay09a] Rolf Haynberg. Der Gotoh-Algorithmus, November 2009. Dokumentation des Algorithmus für das Praktikum Algorithmen der Bioinformatik.
- [Hay09b] Rolf Haynberg. Der Smith-Waterman-Algorithmus, November 2009. Dokumentation des Algorithmus für das Praktikum Algorithmen der Bioinformatik.
- [Wik09a] Wikipedia Contributors. Dynamische Programmierung - Wikipedia. [http://de.wikipedia.org/wiki/Dynamische\\_Programmierung](http://de.wikipedia.org/wiki/Dynamische_Programmierung), November 2009. Letzter Zugriff 19. November 2009.

[Wik09b] Wikipedia Contributors. Optimalitätsprinzip von Bellman - Wikipedia. [http://de.wikipedia.org/wiki/Optimalittsprinzip\\_von\\_Bellman](http://de.wikipedia.org/wiki/Optimalittsprinzip_von_Bellman), November 2009. Letzter Zugriff 19. November 2009.