

Der Waterman-Smith-Beyer-Algorithmus

Praktikum Algorithmen in der Bioinformatik

Rolf Haynberg - 65109948

1 Motivation und Hintergrund

Der hier vorgestellte Algorithmus baut auf dem Needleman-Wunsch-Algorithmus auf und erweitert dessen Funktionalität. Deswegen werden zum Verständnis der folgenden Ausführungen Vorkenntnisse zur Funktionsweise des Needleman-Wunsch-Algorithmus empfohlen. Informationen dazu sind z.B. unter [Hay09b, Heu06, Gus97] zu finden.

Aus dem Needleman-Wunsch-Algorithmus sind uns bereits die Kostenfunktionen *insert*, *delete* und *replacematch* bekannt. Als Eingabe erhalten diese Funktionen das zu löschende oder einzufügende Zeichen bzw. *replacematch* die zu matchenden oder zu ersetzenden Zeichen. Sie erlaubten uns somit eine gewisse Freiheit in der Gewichtung der Distanz ohne signifikante Erhöhung der Laufzeit.

Der Waterman-Smith-Beyer-Algorithmus trägt diesen Gedanken weiter und wird uns erlauben die Kosten für eine Operationen, zu einem gewissen Grad, auch von *vergangenen* Operationen abhängig zu machen. Diese Erweiterung der Funktionalität ist in der Biologie von großer Bedeutung weil damit das Konzept der Gaps in der Berechnung berücksichtigt werden kann.

In den folgenden Abschnitten stellen wir kurz das Konzept des Alignments und das der Gaps vor. Ausführlichere Informationen dazu können z.B. in [Gus97] gefunden werden.

1.1 Alignment

Das Konzept des Alignments ist eng mit dem der Editier-Vorschrift verknüpft (vgl. [Gus97, S. 217]). Während die Editier-Vorschrift anzeigt wie ein String S_1 in einen anderen String S_2 überführt werden kann, macht das Alignment deutlich, an welchen Stellen die Strings übereinstimmen und wo sie sich unterscheiden. Beide Konzepte sind Darstellungen der selben Information und können infolge dessen ineinander überführt werden.

Ein Alignment ist im wesentlichen zwei übereinander geschriebene Strings S_1 und S_2 . An gewissen Stellen sind S_1 und S_2 mit Platzhaltersymbolen gestopft, sodass möglichst wenige verschiedene Zeichen übereinander stehen oder Platzhaltersymbole gebraucht werden.

Beispiel:

$S_1 = \text{R A - D L A G E R}$

$S_2 = \text{W A N D - E R E R}$

Dieses Beispiel ist nur ein mögliches Alignment. Die entsprechende Editier-Vorschrift lautet **replace, match, insert, match, delete, replace, replace, match, match**

Obwohl Alignments für viele Anwendungen, insbesondere in der Bioinformatik, die geeignetere Darstellung ist, werden wir in dieser Dokumentation hauptsächlich Editier-Vorschriften verwenden weil dies unserer Ansicht nach die Funktionsweise des Algorithmus verständlicher macht. Nur im folgenden Absatz über *Gaps* werden wir die Alignment-Darstellung verwenden, da diese den Begriff intuitiver macht.

1.2 Gaps

In einer Vielzahl von Anwendungen des Sequenzalignments, insbesondere aus der Biologie, ist es wahrscheinlicher, dass eine zusammenhängende Teilsequenz aus einer größeren Sequenz gelöscht oder in eine andere Sequenz eingefügt wird, als ihre sukzessive Löschung in mehreren Iterationen. Ein Beispiel dafür sind Mutationen von DNS-Sequenzen. Wird eine auf diese Art und Weise modifizierte Sequenz in einem Alignment ihrer ursprünglichen Form gegenüber gestellt, würden die eingefügten oder gelöschten Teilsequenzen als eine zusammenhängende Kette von Platzhaltern dargestellt werden. Diese maximalen, zusammenhängenden Ketten von Platzhaltern in einem der Strings eines Alignments nennt man *Gaps*.

Für evolutionsgeschichtliche Untersuchungen ist es z.B. sinnvoll *Gaps* zu erkennen da diese etwas über die evolutionsgeschichtliche Nähe der untersuchten Sequenzen verraten [Gus97, S. 236]. Jedoch haben Verfahren wie der Needleman-Wunsch-Algorithmus keinen Mechanismus um *Gaps* zu erkennen. Es würde z.B. ein Alignment mit vielen kleinen *Gaps* ausgegeben werden statt einem, möglicherweise biologisch viel sinnvollerem Alignment mit langen *Gaps*, wenn es eine nur minimal geringere Editier-Distanz hat.

Der Waterman-Smith-Beyer-Algorithmus hingegen erlaubt es, *Gap*-Bildungen zu bevorzugen weil er *Gaps* erkennen kann. Das wiederum liegt daran, dass der Algorithmus in jedem Ausführungsschritt Zugriff auf die bisherigen Alignment-Kandidaten hat.

2 Der Algorithmus

Wie bereits erwähnt, baut der Waterman-Smith-Beyer-Algorithmus auf dem Needleman-Wunsch-Algorithmus auf. Im wesentlichen werden die Kostenfunktionen *insert* und *delete* durch eine einzige *Gap*-Penalty-Funktion $g : \mathbb{N} \rightarrow \mathbb{R}_+$ ersetzt. Die *Gap*-Penalty-Funktion erhält als Eingabe die Länge eines *Gap*s und gibt die Gesamtkosten für das *Gap* zurück. Im speziellen Fall, dass g linear ist, sind also die Kosten für jeden Platzhalter im *Gap* gleich. Dies entspräche den Kostenfunktionen des Needleman-Wunsch-Algorithmus mit zeichenu-nabhängigen Kosten. Sinnvoller ist es jedoch, wenn wir für g eine sublineare Funktion einsetzen. In dem Fall sinken die Kosten für eine Erweiterung eines *Gap*s in Abhängigkeit von der Länge des *Gap*s. Andersherum betrachtet bedeutet dies, dass ein neues *Gap*s zu beginnen teurer ist als ein bestehendes zu erweitern.

Da die *Gap*-Penalty-Funktion nur noch von der *Gap*-Länge abhängig ist, können die Kosten für Einfüge- und Löschoptionen nicht mehr, wie beim Needleman-Wunsch-Algorithmus, in Abhängigkeit des einzufügenden oder zu löschenden Zeichens gewählt werden. Für Match- und Ersetzung-Operationen bleibt diese Möglichkeit aber bestehen.

Die Berechnungsvorschrift einer Zelle der Matrix D lautet:

$$D_{0,0} = 0 \quad (1)$$

$$D_{k,0} = g(k) \quad 1 \leq k \leq n \quad (2)$$

$$D_{0,k} = g(k) \quad 1 \leq k \leq m \quad (3)$$

$$D_{i,j} = \min_k \begin{cases} D_{i,j-k} + g(k) & j \geq k \geq 1 & (4a) \\ D_{i-k,j} + g(k) & i \geq k \geq 1 & (4b) \\ D_{i-1,j-1} + \text{replacematch}(S_1[i], S_2[j]) & i, j \geq 1 & (4c) \end{cases}$$

Für eine Zelle (i, j) der Matrix wird das Minimum der Fälle *Einfügen*, *Löschen* sowie *Ersetzen* bzw. *Match* gesucht unter Berücksichtigung *aller möglichen* Werte k für die Gap-Länge.

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. Darüber hinaus sind bereits die Werte für $D_{i,j}$ mit $i = 0$ (Gleichung 3) oder $j = 0$ (Gleichung 2) angegeben. Dies ist möglich da die Berechnung dieser Werte unabhängig von der Eingabe ist: Für die Berechnung der Werte in Zeile 0 ($i = 0$) trifft immer nur Fall (4a) zu, da in den anderen Fällen die Bedingungen nicht erfüllt sind. Analog dazu trifft für die Berechnung der Werte in Spalte 0 immer nur Fall (4b) zu da dort $j = 0$ ist.

Stellt sich bei der Berechnung des Wertes $D_{i,j}$ heraus, dass es am günstigsten ist, ein Gap in S_1 auf Länge k zu erweitern, dann ergeben sich die Gesamtkosten aus den Kosten bis zum Beginn des Gaps (das ist $D_{i,j-k}$) und den Kosten $g(k)$ für das Gap der Länge k .

Es ist wichtig alle möglichen k Werte zu betrachten, da sich Gaps erst ab einer bestimmten Länge *lohnen* können d.h. geringere Kosten haben als eine Reihe von Replacematch-Operationen und kleineren Gaps.

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 Gap-Penalty Funktion sei

$$g(k) = \sqrt{k}$$

und die Kostenfunktion für Ersetzungs- bzw. Matchoperation sei

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

Aus der Definition (Gleichung ??) können wir bereits die erste Zeile und erste Spalte ablesen. D.h. $D_{i,j}$ für $i = 0$ oder $j = 0$. Damit ergibt sich die Matrix in Abbildung 1.

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

Abbildung 1: Die Editier-Matrix D zu den Zeichenketten $S_1 = \text{WURZEL}$ und $S_2 = \text{VIERTEL}$. Die erste Zeile und erste Spalte geben die Kosten für eine Reihe von Löschr bzw. Einfüge Operationen (Gaps) an. Daran lässt sich gut erkennen wie die Kosten entsprechend der Gap-Penalty-Funktion nur sublinear steigen. Zur Übersichtlichkeit sind die Zeichenketten an den Rändern nochmals aufgetragen.

In diesem Beispiel füllen wir die Matrix anti-diagonal aus. Für $D_{1,1}$ berechnen wir

$$D_{1,1} = \min\{1 + g(1), 1 + g(1), 0 + 1\} = 1.$$

Bereits nach ein paar Schritten ergibt sich das gesuchte Minimum aus fünf Werten. In Abbildung 2 ist die Abhängigkeit des Wertes in Zelle $(2, 2)$ dargestellt. Der Wert $D_{2,2} = 2$

ergibt sich aus

$$D_{2,2} = \min \begin{cases} D_{2,0} + g(2) = 2,8 \\ D_{2,1} + g(1) = 3,0 \\ D_{0,2} + g(2) = 2,8 \\ D_{1,2} + g(1) = 3,0 \\ D_{1,1} + 1 = 2,0 \end{cases}$$

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

Abbildung 2: Der Wert in Zelle (2,2) hängt von allen Zellen 'darüber' und 'davor' ab. Je weiter der Algorithmus fortschreitet, desto größer wird die Zahl der Vergleiche pro Berechnungs-Schritt.

4 Zusammenfassung

Der Waterman-Smith-Beyer Algorithmus erlaubt es mit Hilfe von Gap-Penalty-Funktionen Gap-Bildungen in Alignments zu bevorzugen. Da die Definition ohne genauere Angaben zu der Penalty-Funktion auskommt, bietet er eine hohe Freiheit in der Wahl der Funktion. Jedoch steht dem Flexibilitätsgewinn eine signifikant erhöhte Laufzeit gegenüber. Letzteres ist auch der Grund weshalb der Waterman-Smith-Beyer Algorithmus für die Praxis ungeeignet ist und kaum Einsatz findet.

Schränkt man die Gap-Penalty-Funktion auf affine Funktionen ein, lässt sich der Gotoh-Algorithmus verwenden und die Laufzeit auf $O(nm)$ absenken (siehe auch [Hay09a]).

Laufzeitanalyse

Um den Wert $D_{i,j}$ einer Zelle (i, j) zu berechnen müssen $i + j + 1$ Zellen, nämlich $(i - k, j)$, $(i, j - k)$ und $(i - 1, j - 1)$, untersucht werden. Diese Operationen sind in diesem Algorithmus unvermeidbar da sie für die Berechnung der Gap-Penalty-Funktion $g(k)$ vorliegen müssen. Die Laufzeit lässt sich also schreiben als

$$\sum_{i=0}^n \sum_{j=0}^m i + j + 1 = nm + n \left(\sum_{j=0}^m j \right) + m \left(\sum_{i=0}^n i \right)$$

Da $(\sum_{i=0}^n i) \in O(n^2)$ ist, liegt der Algorithmus in $O(mn^2 + m^2n)$. Für $n \approx m$ ergibt sich auch $O(n^3)$.

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 Needleman-Wunsch-Algorithmus. November 2009. Dokumentation des Algorithmus für das Praktikum Algorithmen der Bioinformatik.
- [Heu06] Volker Heun. Skriptum zur Vorlesung Algorithmische Bioinformatik I/II, 2005-2006. gehalten im SS 05 und WS 05/06.