

Der Gotoh-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. Des weiteren sind zum Verständnis der folgenden Ausführungen Vorkenntnisse zum Waterman-Smith-Beyer Algorithmus sowie Gaps von Vorteil. Weitere Informationen zum Needleman-Wunsch-Algorithmus sind in z.B. unter [Hay09a, Heu06, Gus97] zu finden und Informationen zum Waterman-Smith-Beyer-Algorithmus sowie zu Gaps sind z.B. unter [WSB76, Hay09b] zu finden.

Es gibt eine Vielzahl von Anwendungen in denen es wünschenswert sein kann, Gap-Bildungen in Alignments zu bevorzugen (vgl. [Hay09b][Gus97, Absch. 11.8.2, S. 236]). Der Waterman-Smith-Beyer-Algorithmus ermöglicht dies durch Gap-Penalty-Funktionen. Dabei macht der Algorithmus keinerlei Vorgaben an die Ausprägung der Penalty-Funktionen (lediglich vernünftige Annahmen fordern eine sublineare Funktion). Diese Freiheit schlägt sich jedoch in dessen Laufzeit von $O(nm^2 + mn^2)$ nieder, weshalb der Algorithmus in der Praxis kaum Anwendung findet.

Der Gotoh-Algorithmus begrenzt die Gap-Penalties auf affine Funktionen und ermöglicht so Gap-Bildungen zu bevorzugen und Laufzeiten in $O(nm)$.

Affine Funktionen

Affine Funktionen werden auch als *lineare Funktionen* oder *Geradengleichungen* bezeichnet. Eine affine Funktion $g : \mathbb{R} \rightarrow \mathbb{R}$ lässt sich immer in der Form $g(x) = ax + b$ schreiben. In unserem Fall ist x die Länge des betrachteten Gaps und der Funktionswert sind die Gesamtkosten des Gaps. Man kann daher b als die Kosten für das öffnen eines Gaps interpretieren und a als die Kosten um ein Gap fortzuführen. Durch geschickte Wahl der Parameter a und b können Gap-Bildungen bevorzugt werden, da lange Gaps günstiger werden als z.B. mehrere kurze. Dies liegt daran, dass bei einem großen Gap im Gegensatz zu mehreren kleinen Gaps nur einmalig die Kosten b in die Berechnung eingehen.

2 Der Algorithmus

Als Eingabe erhält der Algorithmus zwei Zeichenketten S_1 und S_2 mit Länge n bzw. m , zwischen denen das optimale Alignment 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 Symbol \perp , welches sonst nicht im verwendeten Alphabet auftaucht.

Des weiteren greift der Algorithmus auf Kostenfunktionen zu, welche die Kosten (oder Gewichte) für jede mögliche Editier-Operation definieren (genauer wird weiter unten beschrieben).

Gesucht ist die Editier-Vorschrift mit den niedrigsten Kosten um S_1 in S_2 zu überführen unter Berücksichtigung von affinen Gap-Kosten.

Wie beim Needleman-Wunsch-Algorithmus wird dynamisch eine $(n + 1) \times (m + 1)$ Matrix D erstellt, sodass $D_{i,j}$ die Editier-Distanz zwischen $S_1[1..i]$ und $S_2[1..j]$ ist. Die Editier-Distanz zwischen S_1 und S_2 ist demnach $D_{n,m}$.

Wir erinnern uns, dass beim Needleman-Wunsch-Algorithmus der Wert einer Zelle das Minimum aus den drei Fällen *Einfügen*, *Löschen* und *Replace bzw. Match* ist. In jeder Zelle wird dann für das Traceback gespeichert durch welche Operationen sich das Minimum bilden lässt. Die Werte, die größer sind als das Minimum werden nicht gespeichert.

Die dem Gotoh-Algorithmus zugrunde liegende Idee ist es, pro Zelle die Kosten für *jeden* der drei Fällen zu speichern. Dadurch ist es möglich, bei der Berechnung einer neuen Zelle Gap-Fortführung von Gap-Beginn zu unterscheiden. Dennoch benötigt der Algorithmus zur Berechnung einer Zelle der Kosten-Matrix wie der Needleman-Wunsch-Algorithmus nur drei unmittelbar benachbarte Zellen.

Es liegen also pro Zelle drei Werte vor. Diese können als Tripel interpretiert werden, sodass D zu einer Matrix aus Tripeln wird. Im Folgenden werden wir jedoch eine andere Darstellung wählen. Wir verwenden drei $(n + 1) \times (m + 1)$ Matrizen E , L und D (diese Darstellung ist auch üblich in der Literatur, vgl. [Gus97]). Die Matrix E enthält die Kosten die sich durch einfügen eines Zeichens an der entsprechenden Stelle in S_1 ergeben. L enthält die Kosten die sich durch das Löschen eines Zeichens ergeben und D enthält, genau wie beim Needleman-Wunsch-Algorithmus, das Minimum von Einfüge-, Lösch- und Replacematch-Operation.

Die Initial-Werte werden wie folgt gewählt:

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

Die weiteren Werte Berechnen sich folgendermaßen:

$$E_{i,j} = \min \begin{cases} D_{i,j-1} + a + b & j \geq 1 \\ E_{i,j-1} + a & j \geq 2 \end{cases} \tag{2a}$$

$$E_{i,j} = \min \begin{cases} D_{i,j-1} + a + b & j \geq 1 \\ E_{i,j-1} + a & j \geq 2 \end{cases} \tag{2b}$$

$$L_{i,j} = \min \begin{cases} D_{i-1,j} + a + b & i \geq 1 \\ L_{i-1,j} + a & i \geq 2 \end{cases} \tag{3a}$$

$$L_{i,j} = \min \begin{cases} D_{i-1,j} + a + b & i \geq 1 \\ L_{i-1,j} + a & i \geq 2 \end{cases} \tag{3b}$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + \text{replacematch} & i, j \geq 1 \\ E_{i,j} & \\ L_{i,j} & \end{cases} \tag{4a}$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + \text{replacematch} & i, j \geq 1 \\ E_{i,j} & \\ L_{i,j} & \end{cases} \tag{4b}$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + \text{replacematch} & i, j \geq 1 \\ E_{i,j} & \\ L_{i,j} & \end{cases} \tag{4c}$$

Dabei ergeben sich a und b aus der Definition der Penalty-Funktion: $g(x) = ax + b$

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 enthalten die obigen Berechnungsvorschriften Bedingungen. Diese sorgen dafür, dass für die Berechnung nur auf definierte Werte zurückgegriffen wird. Bei genauerer Betrachtung fällt auf, dass E und L keine Berechnungsvorschriften für $j = 0$ bzw. $i = 0$ enthalten. In E ist also die erste Spalte undefiniert und in L die erste Zeile. Dies lässt sich verstehen, wenn man sich klar macht, welche Bedeutung die Zellen haben. Die erste Spalte in E steht für die Kosten eines eingefügten Zeichens vor S_1 , also noch bevor der Vergleich mit dem ersten Zeichen von S_2 gemacht wird. Eine solche Operation wird nie die Gesamtkosten verringern und widerspricht damit der Funktionsweise des Algorithmus und ist daher auch nicht definiert. Ähnlich verhält es sich mit der erste Zeile in L .

In der Literatur sind auch Definitionen zu finden, die eine Definitionslücke umgehen indem für die betroffenen Zellen der Wert ∞ angenommen wird. Dadurch findet der Fall bei der Minimumsbildung nie Berücksichtigung was zum selben Effekt führt.

Da die Werte in der Berechnungsvorschrift zu $D_{i,j}$ von $E_{i,j}$ und $L_{i,j}$ abhängen und umgekehrt die Werte in E und L von vergangen Werten aus D , kann die Reihenfolge bei der Berechnung der Werte nicht frei gewählt werden.

3 Beispiel

Betrachten wir nun ein konkretes Beispiel mit

$$S_1 = \text{S I E}$$

$$S_2 = \text{S A H N E}$$

Die affine Gap-Penalty-Funktion sei

$$g(k) = 21 + k \cdot 1$$

D.h. die Kosten beim Öffnen eines neuen Gaps sind 1 und für eine Weiterführung ebenfalls 1.

Die Kostenfunktion für Ersetzungs- bzw. Match-Operation sei

$$\text{replacematch}(x, y) = \begin{cases} 0 & \text{falls } x = y \\ 2 & \text{sonst} \end{cases}$$

Aus der Definition (Gleichung 1) können wir bereits $D_{0,0} = 0$ ablesen.

Für die Berechnung der anderen Werte in D müssen zunächst die entsprechenden Werte in E und L berechnet werden.

Für den Wert $E_{0,1}$ trifft nur Gleichung (2a) zu, da in dem anderen Fall (2b) die Bedingung nicht erfüllt ist (denn $j < 2$). Wir finden also $E_{0,1} = 2$. Ähnlich dazu lässt sich $L_{1,0} = 2$ berechnen. Mit diesen Werten können nun $D_{1,0}$ und $D_{0,1}$ berechnet werden. Setzt man dies fort, kann die erste Zeile in E , die erste Spalte in L sowie die erste Zeile und erste Spalte in D berechnet werden.

Für Zelle $E_{1,1}$ ist in Gleichung 2 wieder nur Fall 2a definiert. Dadurch ergibt sich der Wert 4. Ähnliches gilt für Zelle $L_{1,1}$. Hier finden wir auch Wert 4. Zur Berechnung des Wertes $D_{1,1}$ muss daher das Minimum aus 4, 4 und 0 gefunden werden (die 0 ergibt sich weil hier ein Match vorliegt und $D_{0,0} = 0$ ist). Also ist $D_{1,1} = 0$. Weitere Schritte sind in Abbildung 1 dargestellt.

\mathcal{E}	\perp	S	A	H	N	E	\mathcal{L}	\perp	S	A	H	N	E	\mathcal{D}	\perp	S	A	H	N	E
\perp	<i>n.d.</i>	2	3	4	5	6	\perp	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>	\perp	0	2	3	4	5	6
S	<i>n.d.</i>	4	2	3			S	2	4	5	6	7	8	S	2	0	2	3		
I	<i>n.d.</i>	5	4				I	3	2	4				I	3	2	2			
E	<i>n.d.</i>	6					E	4	3					E	4	3				

Abbildung 1: Von links nach rechts: Die Matrizen E , L und D zu den Zeichenketten $S_1 = \text{SIE}$ und $S_2 = \text{SAHNE}$. Die erste Spalte in E ist undefiniert ebenso wie die erste Zeile in L . Die Kosten für das erste Gap betragen 2. Jedes angeschlossene Gap erhöht die Kosten aber nur noch um 1. Der Wert $D_{1,1} = 0$ ist das Minimum aus $E_{1,1}$, $L_{1,1}$ und die Kosten für ein Match.

Setzt man das Beispiel weiter fort, so ergeben sich die Alignments

S I - - E und S - - I E

S A H N E und S A H N E

Wenn jedoch lineare Gap-Kosten gewählt werden, z.B. $g(k) = 0 + k \cdot 1$, dann unterscheidet sich der Algorithmus im Ergebnis nicht von dem des Needleman-Wunsch-Algorithmus.

In diesem Beispiel ergibt sich dann eine weitere Lösung

S - I - E

S A H N E

Dieses Alignment ist keine Lösung in dem behandelten Beispiel denn jetzt liegen zwei Gaps vor. Die Kosten für zwei Gaps sind im Falle einer nicht-linearen Gap-Kostenfunktion höher als die Kosten für ein zusammenhängendes der gleichen Länge. Damit sind die Gesamtkosten nicht mehr optimal und stellen auch keine Lösung dar. Also zeigt der Algorithmus in diesem Beispiel das gewünschte Verhalten, nämlich Gap-Bildungen zu bevorzugen.

4 Zusammenfassung

Der Gotoh-Algorithmus erlaubt es Gap-Bildungen mittels affinen Gap-Panalties zu bevorzugen. Dabei bietet er die gleiche Laufzeitkomplexität des Needleman-Wunsch-Algorithmus. Nachteile können zum einen im erhöhten Speicherverbrauch gesehen werden. Er benötigt etwa drei mal mehr Speicher als der Needleman-Wunsch-Algorithmus weil nun drei Matrizen statt einer Matrix gespeichert werden müssen. Die Ausmaße dieses Nachteils lassen sich jedoch mindern indem Modifikationen wie beim Hirschberg-Algorithmus vorgenommen werden. Damit lässt sich eine lineare Platzkomplexität erreichen. Ein weiterer Nachteil ist, dass er in der Wahl der Gap-Panalty-Funktion nicht die Flexibilität des Waterman-Smith-Beyer-Algorithmus bieten kann sondern auf affine Funktionen begrenzt ist. Seine Stärke kann deshalb in dem Kompromiss aus beiden Ansätze gesehen werden.

Laufzeitanalyse

Um den Wert einer Zelle $X_{i,j}$ zu berechnen müssen die drei Nachbarzellen $X_{i-1,j}$, $X_{i,j-1}$, $X_{i-1,j-1}$ untersucht werden für jedes $X \in \{E, L, D\}$. Außerdem müssen für den letzten Fall die Strings an den Positionen i bzw. j untersucht werden. Diese Kosten können als konstant angenommen werden. 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 Needleman-Wunsch-Algorithmus. November 2009. Dokumentation des Algorithmus für das Praktikum Algorithmen der Bioinformatik.
- [Hay09b] Rolf Haynberg. Der Waterman-Smith-Beyer-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.
- [WSB76] M. S. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367 – 387, 1976.