

Praktikum Data-Mining und Sequenzanalyse

Alignments

M. Meusel, B. Vogel

Lehrstuhl für Bioinformatik
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena

1. Dezember 2014

Programm Heute

- Motivation
- Problem
- Algorithmen
 - naives Verfahren
 - Needleman-Wunsch-Algorithmus
 - Smith-Waterman-Algorithmus
 - Hirschberg Algorithmus
- Implementierung
- Aufgaben

Motivation

- Identifizierung konservierter Muster
- Entdecken evolutionärer Beziehungen
- Untersuchung unbekannter Proteine auf Funktion und Struktur

Problem

Ein *Sequenzalignment* A zweier Sequenzen a und b über Alphabet Σ ist ein Paar von Wörtern a^\diamond und b^\diamond gleicher Länge über dem Alphabet $\Sigma \cup \{-\}$, so dass aus a^\diamond und b^\diamond durch Löschen der Gapsymbole a und b entsteht und a^\diamond und b^\diamond keine Gapsymbole $-$ an gleicher Position enthalten.

Ein Alignment zweier Teilsequenzen von a und b heißt *lokal für a und b* .

Problem

Ein *linearer Alignment score* ist eine Funktion auf Alignments $A = (a^\diamond, b^\diamond)$ der Form

$$\sigma(A) = \sum_{1 \leq i \leq |A|} \sigma(a_i^\diamond, b_i^\diamond)$$

Problem

Gegeben sei eine Bewertungsfunktion σ für Alignments über Σ . Das *Alignmentproblem* besteht darin für zwei Sequenzen über Σ das durch σ am besten bewertete Alignment (bzw. dessen Bewertung) zu finden.

Naiver Algorithmus

- Finde bestes (globales) Alignment durch *vollständige Aufzählung* aller Alignments.
- Wie können alle Alignments von a und b aufgezählt werden?
 - rekursiver Aufbau der Alignmentsequenzen a^\diamond und b^\diamond
 - dabei “Fortschreiten” in a und b (Positionszeiger).
- 3 rekursive Aufrufe in der Funktion:
 - Aligniere aktuelle Zeichen von a und b
 - Aligniere aktuelles Zeichen von a mit Gap
 - Aligniere aktuelles Zeichen von b mit Gap

Naiver Algorithmus lokal

- Wie bekommt man das *beste lokale Alignment* bei Aufzählung aller globalen Alignments (weniger naiv)?
- bestes lokales Alignment ist optimales Teilalignment eines globalen Alignments
- Suche bestes Teilalignment in linearer Zeit
 - Maximale Teilsumme
 - Scanning line-Prinzip

Naiv aber iterativ?

- `while`-Schleife statt rekursiver Aufruf
- Counter für Rekursionstiefe
- Arrays für Variablen, die beim Aufruf übergeben werden
- Array für Position in der aufrufenden Funktion

Needleman-Wunsch Alignment

DP-Algorithmus zur Berechnung eines globalen Alignments.

- Matrix D mit $n \times m$ Einträgen
- Jeder Matrixeintrag $D(i, j)$ entspricht der besten Bewertung eines Teilalignments von a_1, a_2, \dots, a_i und b_1, b_2, \dots, b_j
- Bewertungsfunktion (Scoring) $\sigma(x, y)$ bewertet zwei Symbole x und y .

Needleman-Wunsch Rekurrenz

$$D(i, j) = \max \begin{cases} D(i-1, j-1) & + \sigma(\mathbf{a}_i, \mathbf{b}_j) \\ D(i-1, j) & + \sigma(\mathbf{a}_i, -) \\ D(i, j-1) & + \sigma(-, \mathbf{b}_j) \end{cases}$$

Needleman-Wunsch Initialisierung

- $D(0, 0) = 0$
- Initialisierung der zusätzlichen (ersten) Spalte und (ersten) Zeile:
 - $D(i, 0) = D(i - 1, 0) + \sigma(a_i, -); i = 1, 2, \dots, n - 1$
 - $D(0, j) = D(0, j - 1) + \sigma(-, b_j); j = 1, 2, \dots, m - 1$

Nun kann zeilenweise oder spaltenweise ab $i = 1$ und $j = 1$ die Rekurrenz angewandt werden.

Needleman-Wunsch Traceback

Berechnung des Alignments aus der gefüllten Matrix D

- Start an $D(n, m)$ und berechnung rückwärts bis zum Eintrag $D(0, 0)$. Aus welcher Richtung kommt $D(n, m)$? Je nach Richtung bzw. Fall baut man das Alignment von hinten entsprechend auf, d.h man aligniert zwei Buchstaben oder fügt ein gap ein.
- Parallele Berechnung einer Traceback Matrix T .

Smith-Waterman Alignment

- Es wird kein globales Alignment, sondern ein lokales Alignment berechnet, d.h. es wird das beste Alignment von Substrings aus n und m gefunden.
- Beim Berechnen der Matrix gibt es keine negativen Werte.
- Das Traceback beginnt nicht bei $D(n, m)$, sondern bei $\max_{i,j}\{D(i, j)\}$, also dem größten Wert der Matrix; es endet, sobald man auf eine Zelle mit Inhalt 0 stößt.

Smith-Waterman Rekurrenz

$$D(i, j) = \max \begin{cases} 0 \\ D(i-1, j-1) + \sigma(a_i, b_j) \\ D(i-1, j) + \sigma(a_i, -) \\ D(i, j-1) + \sigma(-, b_j) \end{cases}$$

Linearer Speicherverbrauch

- Bei quadratischem Speicherbedarf:
 - Alignment von zwei Virus-Genomen zu je *2000 Basen*
⇒ ca. $4 \cdot 2000^2 = 16 \text{ Megabyte}$
 - Alignment von zwei Bakterien-Genomen zu je *1 Mio. Basen*
⇒ ca. $4 \cdot (10^6)^2 = 4 \text{ Terabyte}$
- Idee:
 - es genügt, statt kompletter Matrix die letzte Zeile zu speichern
 - Zeile *i in-place* aus Zeile *i – 1* berechenbar
 - Hilfsvariable für schräg oben
 - Folge: Traceback nicht mehr aus DP-Matrix berechenbar

Hirschberg Algorithmus

- algorithmisches Prinzip: *Divide-And-Conquer*
- Berechne Index-Matrix M für eine Zeile m'

$$M(i, j) = \begin{cases} \text{undef.} & | i < m' \\ j & | i = m' \\ M(i-1, j) & | D(i, j) = D(i-1, j) + \text{gap} \\ M(i-1, j-1) & | D(i, j) = D(i-1, j-1) + \text{score} \\ M(i, j-1) & | D(i, j) = D(i, j-1) + \text{gap} \end{cases}$$

- Rekursion für kürzere überlappende Teilsequenzen
- globales Array der Durchstoßungspunkte $C(i)$

Implementierung I

- Klasse `Alignment` soll ein `Alignment`, d.h. zwei Sequenzen **speichern**.
- Klasse `Evaluator` implementiert die Bewertungsfunktion und kann ein `Alignment` bewerten. Dazu verwaltet die Klasse eine Scoringmatrix (`int`) und besitzt auch eine Methode, um diese auszulesen. Die Scoringmatrix soll aus einer Datei einlesbar sein.
- Eine Klasse `Aligner` berechnet optimale `Alignments`. Es soll auch möglich sein Bewertungen ohne den zusätzlichen Aufwand des Tracebacks zu berechnen.

Implementierung II

- Von den Klassen sollen in geeigneter Weise Subklassen gebildet werden, die die verschiedenen Algorithmen implementieren und sowohl globale als auch lokale Alignments berechnen können.
Eventuell ist es sinnvoll, einzelne Superklassen abstrakt zu definieren.
- die Klassen sollen in einem Package `alignments` zusammengefasst werden.

Aufgaben

- Implementierung der Klassen mit den entsprechenden Algorithmen
- API Dokumentation mit javadoc
- Projektmanagement mit Gradle
- Unit-Tests
- Benutzerinterface als Kommandozeilenprogramm (CLI) + README
- Testläufe und Testdaten mit Protokollieren der Ergebnisse (Testdaten und Aufgabenblatt auf der Kurswebsite)