BIOINFORMATICS

Vol. 00 no. 00 2013
Pages 1–2

Algorithmische Bioinformatik - Praktikum 2

Co-Authoren : Maximilian Driller, Philipp Borchert und Pratik Dhakal

FU Berlin

23.12.2013

ABSTRACT

Motivation: Wir versuchen mit uns zur Verfügung gestellten Sequenzen einen Phylogenetischen Baum zu erstellen. Mit Hilfe der MUSCLE Webservice wird eine Distanzmatrix erzeugt und auf der Matrix wird der UPGMA Algorithm implementiert. Danach werden die Sequenzen mit verschiedenen Mutationsrate mutiert und das selbe Verfahren wird auf allen mutierten Sequenzen angewendet.
Der Bericht selbst sollte die Problemstellung, ihre Herangehensweise und die Ergebnisse dokumentieren. Ausserdem soll die Implementierung erläutert werden. Hier geht es nicht darum, sämtliche Variablen und Methoden zu erläutern, sondern darum die Grundidee hinter den einzelnen Klassen zu beschreiben und den Ablauf des Programs (mit Bezug auf die Aufgabenstellung) zu erläutern.
Results: Als Ergebnis erhalten wir verschiedene Phylogentischen Bäumen sowohl fr die original Sequenzen als auch fr die mutierten Sequenzen.
Kontakt: pratik@zedat.fu-berlin.de , PhilippB1990@zedat.fu-berlin.de, mdriller@zedat.fu-berlin.de

1 EINLEITUNG

Anhand immer größerer Datenmengen ist heute eine schnelle Analyse von Sequenzen essentiell in der Bioinformatik. Aus diesem Grund werden Programmiersprachen wie Python,Java etc. benutzt, um eine schnelle Interpretation der Sequenzen und ihr evolutionären Verfahren in Organismen zu ermöglichen. Für dieses Unterfangen haben wir uns für eine Lösung mit der Programmiersprache Python entschieden. Durch viele, auf die Bioinformatik zugeschnittenen Pakete, ist es eine ideale Sprache um dieses Vorhaben umzusetzen.

1.1 Problematik

Bei der Bearbeitung der Aufgaben sind wir auf folgende Probleme gestoßen:

1. Bearbeitung der Matrizen
   Lösung: Da in Python keine Matrix Struktur vorhanden ist, haben wir erst versucht über Python Paket Numpy mit Matrix Struktur zu arbeiten. Bei dieser Umweg hatten wir Probleme beim Verändern der Werte in der Matrix. Deswegen haben wir auf Matrix-struktur bzw. Numpy für dieses Zwecks verzichtet und haben mit Listen gearbeitet. Wir haben Matrix dadurch repräsentiert, in dem wir Listen in Listen geschachtelt haben.

2. Bootstrapping
   Nach Stunden lange Nachforschung sind wir trotzdem auf keine vernünftige Weise zur Implementierung von Bootstrapping in Python gestoßen.

3. Dynamisierung des Erstellen von Bäumen
   Da die Bäume, vor der graphischen Darstellung, erst in Newick format gespeichert werden konnten wir die Erstellung von Newick Format Dateien nicht dynamisieren und mussten daher die Codes für einzeln Bäumen manuell einfügen.

2 IMPLEMENTIERUNG

2.1 Übersicht

Zum Beginn unserer Implemetierung haben wir die benötigten Pakete : RESTService, numpy, Phylo, usw importiert. Als nächstes erfolgte der die Deklaration unserer Path-Variable (Zeile 332), ber die unser Quellverzeichnis angegeben wird. Die Abänderung dieser Variable ist für Ausführung dieser Datei essentiell.

2.1.1 dist_matrix Im weiteren Verlauf (Zeile 346) erfolgt die Ausführung der Funktion 'dist_matrix' durch die Angabe der gegebene Sequenzen als eine Fasta Datei. In der Funktion 'dist_matrix' (Zeile 143) erfolgt erst der Aufruf der MUSCLE webservice, welche als eine Klasse ab Zeile 17 bis Zeile 140 zu finden ist. Durch den MUSCLE Webservice bekommen wir eine Percent Identity Matrix (PIM) zurück.
In der darauf folgenden For-Schleife wird die Percent Identity Matrix zu einer Distanz Matrix umgewandelt. Die Distanz Matrix wird zunchst zu einer oberen Dreiecksmatrix umgewandelt, wobei die Werten in den unteren Dreieck mit '-' ersetzt werden. Diese ist ntig um den UPGMA Algorithm zu implementieren.
Die For-Schleife ab Zeile 197 kümmert sich um die Implementierung des UPGMA Algorithmns. Als erstes wird der kleinste Wert der Distanzmatrix festgestellt, wessen Zellposition in der Distanzmatrix als eine Liste mit Hilfe des Befehls 'print pos_min' (Zeile 215) ausgegeben wird. Wie nach UPGMA benötigt, wird im nächsten Schritt, die Zeile und Spalte der Distanz Matrix mit der kleinsten Wert gelöscht. Dies erfolgt durch die zwei if-else Schleifen.
Nachdem Löschen der Zeilen und Spalten muss wieder eine neue Spalte zu der Distanz Matrix eingefügt werden. Die zu der neuen Spalte gehrenden Werten werden als 'dzw' (Zeile 264) berechnet. Alle diesen Werten werden in einer Liste namens 'dzws' gespeichert und spter zu der Distanz Matrix hingefgt. 'Nx' und 'Ny' bezeichnen die Clusters, welche durch eine Liste 'cluter_references' immer

aktualisiert wird. Da unsere Distanz Matrix immer eine quadratische Matrix sein muss, wird eine neue leere Zeile zu der Matrix hingefgt. So verläuft der Suche nach der kleinsten Wert bzw. der UPGMA Algorithm bis nur einen Wert in der Matrix vorhanden ist. Dieser einen Wert wird am Ende mit der if-Schleife als der letzte kleinste Wert festgestellt.
Als Resultat erhalten wir alle kleinsten Werte und deren Positionen welche zur Konstruktion des Baums notwendig sind.

2.1.2 to_mutate Mit der For-Schleife (Zeile 335) wird dieser Funktion aufgerufen. Die For-Schleife und die Funktionen dienen zur Erstellung von fünf mutierte Sequenzen mit verschiedenen Mutationsraten. Die erste mutierte Sequenzen hat die Mutationsrate von 5%. Die Mutationsrate steigt um 5% bei jede weitere Mutation.

2.1.3 weiterer Verlauf Wir haben beim Testlauf unseres Programms die kleinsten Werten fr jede Sequenzen ausgeben lassen. Mit diesen Werten haben wir fr jede Sequenz manuell ein Baum erstellen knnen, welche als Variable 'b' für jeweiligen Sequenzen in dem untersten Teils des Codes zu finden sind. Diese Bäume haben wir in Newick format als einzelnen Daten gespeichert. Dann mit Hilfe des 'Phylo.read' Befehls haben wir die Bäume in Python graphisch darstellen können.

3 ERGEBNISSE

Wir erhalten als Resultat sechs verschiedene phylogenetischen Bäume, welche den Unterschied durch die Mutation verdeutlichen. Der erste phylogentische Baum mit den Namen der Spezies ist für die Sequenzen ohne Mutation.
Zur Vereinfachung, haben wir in anderen phylogenetischen Bäume die Blätter als Buchstaben ausgeben lassen, welche die folgende Sequenzen repräsentieren.
a A5MFU6_STRPN
b NEUR4_MOUSE
c NEUR4_HUMAN
d NRAM_INBLE
e NRAM_I67A0
f NRAM_I75A5
g NRAM_I63A3
h NRAM_I18A0
i C4Al48_9INFA
j C6KQG3_9INFA

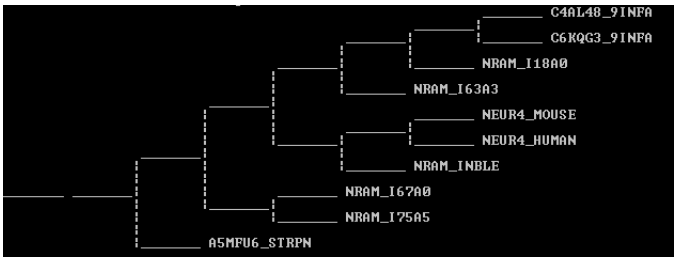

Fig. 1. nicht mutierte Sequenzen

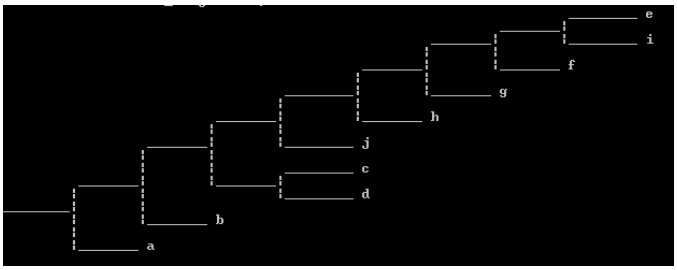

Fig. 2. 5% mutierte Sequezen

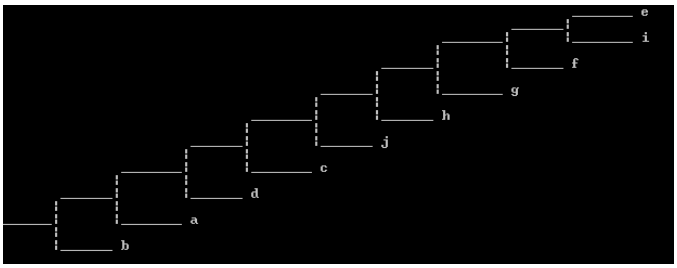

Fig. 3. 10% mutierte Sequenzen

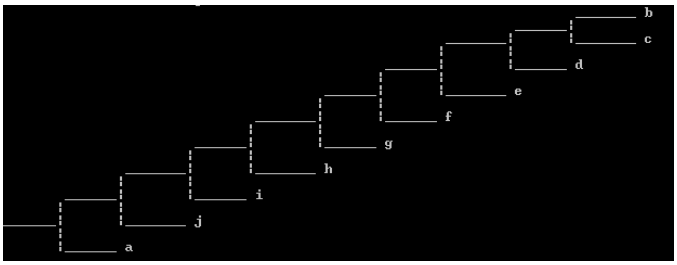

Fig. 4. 15% mutierte Sequenzen

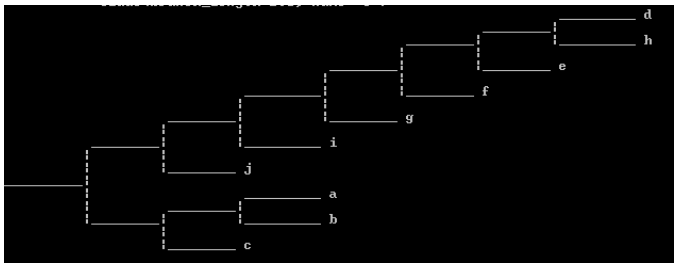

Fig. 5. 20% mutierte Sequenzen

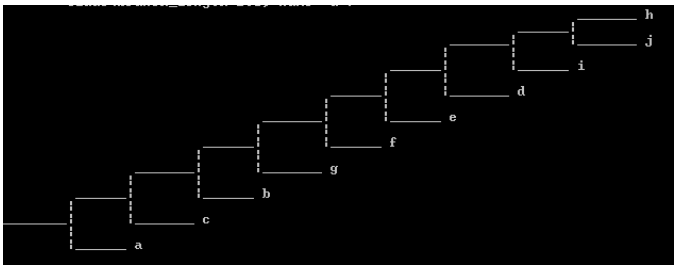

Fig. 6. 25% mutierte Sequenzen