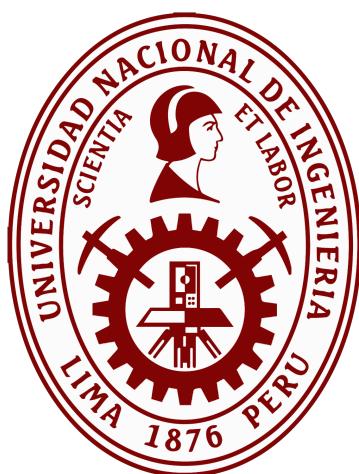


UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE CIENCIAS

CIENCIAS DE LA COMPUTACIÓN



PROYECTO DE BIOLOGIA COMPUTACIONAL

Título del Trabajo

GATTACA

Autores

Lázaro Camasca, Edson Nicks
Leon Rios, Marco Naro

Profesor

Nuñez Iturri, Ciro Javier

Lima - Peru
(2019)

Contents

1	Objetivos	2
1.1	Objetivos Generales	2
1.2	Objetivos Específicos	2
2	Resumen Ejecutivo	2
3	Descripción del Proyecto	2
3.1	Determinar las especies y el material a utilizar	2
3.2	Elegir los marcadores moleculares	3
3.2.1	Proteina NADH deshidrogenasa	3
3.3	Realizar el alineamiento múltiple de genes homólogos	4
3.4	Modelo Evolutivo Kimura	4
3.4.1	Construcción de la matriz de distancias	4
3.5	Métodos para la construcción de árbol filogenético	4
3.5.1	UPGMA	5
3.5.2	Unión de Vecinos	5
4	Algoritmos e implementación computacional	5
4.1	Prerrequisitos	5
4.2	Obtener Secuencias	5
4.3	Union de archivos fasta	6
4.4	Alineamiento de Secuencias	7
4.4.1	Alineamiento Multiple	7
4.5	Lectura de Secuencias Alineadas	7
4.6	Creación de la Matriz de Distancias	8
4.7	Creación del Arbol Filogenetico	8
4.8	Interfaz Grafica	9
5	Resultados	9
5.1	Verificar la fiabilidad del árbol construido	9
5.2	Analizar el árbol filogenético	10
5.2.1	Modelamiento de la Estrutura de Proteinas	10
6	Conclusiones	10
7	Apéndice	10

1 Objetivos

1.1 Objetivos Generales

- Creación de un aplicación gráfica usando BioPython y Tkinter para el análisis de información Genética de Especies endémicas de las regiones del Perú.

1.2 Objetivos Específicos

- Recolectar información genética de especies endémicas.
- Desarrollar la aplicación para el análisis de secuencias.,
- Desarrollar algoritmos para obtener las alineaciones, árboles filogenéticos
- Evaluar el árbol filogenético

2 Resumen Ejecutivo

Se pretender crear una aplicación gráfica conectada a una base de datos con la información genética de las especies endémicas del Perú, el software procesara las secuencias, creará el árbol filogenético, mostrará los resultados y analizará las relaciones evolutivas de las especies escogidas. Se escogió como marcaadores moleculares a la proteína NADH deshidrogenasa subunidad 2 debido la importante **función respiratoria mitocondrial**, hecho por el cual tiene presencia en todas las especies escogidas. Se realizó el alineamiento utilizando software Clustal Omega. Se utilizó el modelo evolutivo Kimura y el método para la construcción de árbol escogido fue un método basado en agrupamiento, de los cuales se implemento el método UPGMA y el método Unión de vecinos(NJ).

3 Descripción del Proyecto

El proyecto esta implementado netamente en el lenguaje Python Las librerías utilizadas serán:

- BioPython para el procesamiento de secuencias
- Tkinter para el entorno gráfico.

El software incluye:

- Dentro de la GUI, se pobre escoger las especies para el análisis posterior.
- Datos reales recolectados de la base de datos de NCBI.
- Algoritmos implementados para el alineamiento de Genes homólogos.
- Algoritmos implementados para el alineamiento de Proteínas.
- Algoritmos para la Generación de Arboles Filogenéticos.

La **metodología** que se implantó para el desarrollo del proyecto es el siguiente:

3.1 Determinar las especies y el material a utilizar

Las especies se escogieron fueron especies endémicas del Perú, especies en **peligro de extinción** en el Perú y **especies representativas** del Perú. Además, se tuvo en cuenta la disponibilidad de material genético puesto que muchas de las especies endémicas poseen una base de datos genética incompleta y algunas no se encuentran codificadas en absoluto.

El material genético se encuentran en la base de datos de NCBI, los nombres son un enlace para poder optener más información en NCBI.

Se escogió las siguientes 11 especies:

- [Tremarctos ornatus](#)

- [Panthera onca](#)
- [Vicugna vicugna](#)
- [Aulacorhynchus huallagae](#)
- [Leopardus jacobita](#)
- [Inia geoffrensis](#)
- [Spheniscus humboldti](#)
- [Vultur gryphus](#)
- [Lama glama](#)
- [Cavia porcellus](#)
- [Platalea ajaja](#)

3.2 Elegir los marcadores moleculares

La elección de los marcadores moleculares es una parte importante porque puede hacer una **gran diferencia** en la obtención de un árbol correcto.

Entre los marcadores moleculares, secuencias de nucleótidos o de proteínas, se optó por utilizar **secuencias de proteínas** por las siguientes razones:

- Como se va estudiar la evolución de grupos de **organismos ampliamente divergentes** se aconseja utilizar secuencias de proteínas.
- Las relaciones filogenéticas que se están analizando están en el **nivel más profundo - bacteriana**, por ello lo más adecuado es usar secuencias de proteínas conservadas.

3.2.1 Proteína NADH deshidrogenasa

La proteína a analizarse sera el NADH deshidrogenasa, también conocido como Complejo I, sub-unidad 2 debido a que se encuentra presente en todas las especies y está codificado.

La proteína escogida cumple una importante **función en la respiración bacteriana y mitocondrial**. Por ende, es posible encontrarla en diversas especies y no es extraño que se haya codificado. Una importante observación es que no todas las especies se encuentran codificadas en la base de datos de NCBI, faltando genes importantes.

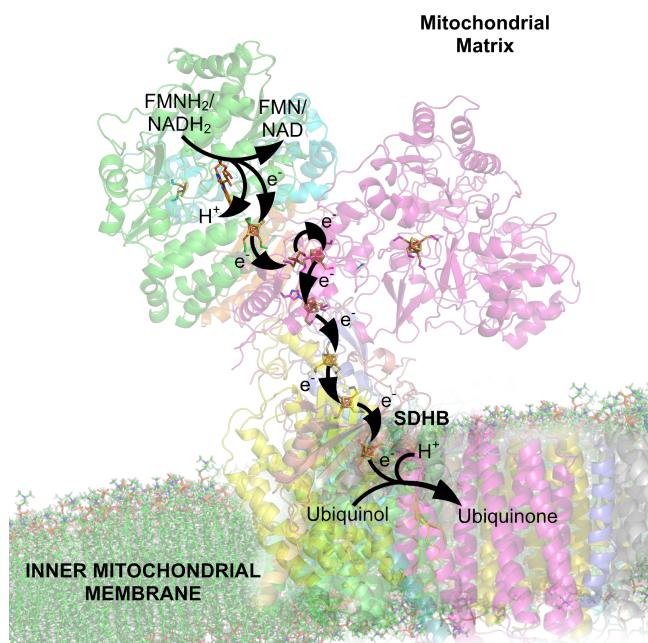


Fig. Proteína NADH deshidrogenasa

3.3 Realizar el alineamiento múltiple de genes homólogos

Este paso es el mas importante de todas, ya que éste establece las correspondencias posicionales en la evolución.

Sólo el alineamiento correcto produce inferencias filogenéticas correctas.

Clustalo Omega

Para el alineamiento de secuencias se utilizo el software de Clustalo Omega. Utilizando el siguiente comando:

```
'clustalo.exe -i filename -o salida --outfmt=clu'
```

Donde en filename se encuentran las secuencias y salida es el archivo con las secuencias alineadas.

El programa [Clustalo Omega](#) se encuentra tanto para windows, Linux y Mac, además se utiliza en mismo comando para realizar el alineamiento en cualquier SO.

EMBL-EBI

En caso que no se quiera descargar el software se puede utilizar la herramienta web de EMBL-EBI [Multiple Sequence Alignment](#).

Donde filename se enviaran a la Web, esta realizara la alineación luego se descargara los resultados en formato clustal.

3.4 Modelo Evolutivo Kimura

Ya que se quiere resultados lo más sofisticado (realista) se opto por el **Modelo Kimura**. Este modelo considera **diferentes las tasas de mutación** para las transiciones (substitución de una purina por otra o una pirimidinas por otra) y para las transversiones (substitución de una purina por una pirimidina o vice versa).

De acuerdo a este modelo las transiciones ocurren más frecuentemente que las transversiones, lo cual provee mejores estimaciones de la distancia evolutiva.

3.4.1 Construcción de la matriz de distancias

A partir del modelo Kimura tenemos:

$$d_{AB} = -\frac{1}{2} \ln(1 - 2p_{ti} - p_{tv}) - \frac{1}{4} \ln(1 - 2p_{tv})$$

Donde:

p_{ti} es la frecuencia observada de transición.

p_{tv} es la frecuencia de transversión.

Las distancias evolutivas calculadas son usadas para construir una matriz de distancias entre todos los pares de taxones.

3.5 Métodos para la construcción de árbol filogenético

Los algoritmos basados en distancias para construir árboles filogenéticos pueden ser subdivididos:

Métodos basados en agrupamiento

Los algoritmos basados en agrupamiento calculan el árbol usando una matriz de distancias e iniciando por los pares de secuencias más similares.

Un gran ventaja es la habilidad para hacer uso de **diferentes modelos de substitución** para corregir las distancias evolutivas.

Métodos basados en optimalidad

Los algoritmos basados en optimalidad comparan muchas topologías alternativas de árboles y seleccionan el que tenga el mejor ajuste entre las distancias estimadas en el árbol y las distancias evolutivas reales

Los métodos basados en optimalidad requieren **muchas capacidades de computo** debido a la búsqueda exhaustiva que realizan, por ello se optó por escoger los métodos basados en agrupamiento.

El usuario podrá escoger dos métodos, a partir de la interfaz gráfica y serán los siguientes:

3.5.1 UPGMA

UPGMA (unweighted pair group method using arithmetic average) El método más simple basado en agrupamiento.

3.5.2 Unión de Vecinos

El método de "Unión de vecinos" parte de una matriz de distancias, que indica la distancia entre cada par de taxones. El algoritmo comienza con un árbol completamente sin resolver, cuya topología corresponde a la de una red en estrella, y aplica los siguientes pasos hasta que el árbol está completamente resuelto y las longitudes de sus ramas.

Ahora que ya sabemos los pasos para la creación de árboles filogenéticos pasamos a la implementación computacional.

4 Algoritmos e implementación computacional

Para poder replicar el proyecto se necesitan ciertos requisitos.

4.1 Prerrequisitos

Los prerrequisitos son los siguientes:

Instalar Python

Instalar python desde [Python](#) para Windows o 'sudo apt-get install python3.6' para Linux.

Instalar BioPython

BioPython es un paquete Python muy popular para manejar información biológica. Para instalar:

- conda install biopython
- pip install biopython

BioPython tiene tres funcionalidades principales: Sequence Handling, 3D Structure, Population Genetics.

Antes de pasar con la implementación necesitamos saber con qué tipo de datos vamos a trabajar, estos datos se encuentran en archivos con terminación .fasta, .xml, .clustal, etc. Para establecer un orden y facilidad en el mantenimiento de los algoritmos estos se encontrarán en una carpeta llamada **data-gen**.

4.2 Obtener Secuencias

Si usted quiere replicar el proyecto con otras especies, lo único que tiene que hacer es descargar sus secuencias en formato .fasta de NCBI y agregarlas al directorio data-gen.fasta, ver la siguiente imagen.

Fig. Descargar la secuencia .fasta haciendo click en [FASTA](#)

BioPython ya tienen implementado algunos algoritmo para el manejo y alineamiento de secuencias, creacion de arboles estos nos facilitan el trabajo.

4.3 Union de archivos fasta

Una vez que tenemos todos nuestros archivos .fasta, pasamos a seleccionar aquellos organismos que nos interasan para su posterior análisis.

Para la selección hacemos uso de una lista llamada elegidos = [] que contiene los indices de las secuencias elegidas. Creamos una funcion que reciba la lista elegidos y crea un archivo en data-gen llamado **Sec-Unidas.fasta**.

```
In [1]: 1 from Bio import SeqIO
2 import os
3
4 def join_Fasta(elegidos):
5     """Une las secuencias que se encuentran en el directorio input_dir
6     y que se encuentren en la lista de los elegidos = [ ]. 
7     Retorna en la secuencia total en el directorio output_dir"""
8
9     input_dir = 'data_gen/fasta/'
10    output_dir = 'data_gen/Sec_Unidas.fasta'
11
12    print('Procediendo a leer los ficheros de ', input_dir)
13    records = []
14    num_seq = -1
15    for seq in os.listdir(input_dir):
16        num_seq += 1
17        if num_seq in elegidos:
18            fichero = open(input_dir + seq)
19            record = SeqIO.read(fichero, 'fasta')
20            records.append(record)
21            SeqIO.write(records, output_dir, 'fasta')
22    print('Se unido todas las secuencias fasta de {} en {}'.format(input_dir, output_dir))
```



```
In [2]: 1 elegidos = [1,2,5,9]
2 join_Fasta(elegidos)
```

Procediendo a leer los ficheros de data_gen/fasta/
Se unido todas las secuencias fasta de data_gen/fasta/ en data_gen/Sec_Unidas.fasta

Ahora que tenemos las secuencias unidas en un solo archivo pasamos a alinearlas.

4.4 Alineamiento de Secuencias

Al alinear dos o más secuencias podemos observar las partes donde ellas difieren e inferir información útil sobre ellas

Un tipo de información útil es la filogenia. Esto implica que podemos inferir cómo la evolución puede explicar un conjunto particular de secuencias observadas

4.4.1 Alineamiento Multiple

Como se esta usando Clustalo Omega en el algoritmo se hara uso de un comando que realize el alineamiento, tambien se pueden encontrar otras herramientas revisando el REEDME que proporciona a la hora de la descarga.

El algoritmo para la implementacion recibe el archivo de las secuencias unidas y genera un archivo **.clustal** con el alineamiento, se hace uso de comandos de sistema.

```
In [3]: 1 import os
2
3 def alinear_Secuencias(filename = 'Sec_Unidas.fasta'):
4     dir_inicio='data_gen'
5     dir_final = 'Clustal_Omega'
6     salida_ali = 'Sec_Alineadas.clustal'
7     print('Alineando las secuencias.....')
8     mover_archivo(dir_inicio, dir_final, filename)
9     #Realizar el alineamiento
10    comando = 'cd Clustal_Omega & clustalo.exe -i {} -o {} --outfmt=clu --force'.format(filename,salida_ali)
11    os.system(comando)
12    print('Se genero el archivo: {}'.format(salida_ali))
13
14    mover_archivo(dir_final, dir_inicio, salida_ali)
15    mover_archivo(dir_final, dir_inicio, filename)
16
17    #-----
18
19    def mover_archivo(dir_inicio, dir_final, filename):
20        comando = 'move {}\\{} {}\\{}'.format(dir_inicio,filename, dir_final, filename)
21        os.system(comando)
22
In [4]: 1 alinear_Secuencias()
Alineando las secuencias.....  
Se genero el archivo: Sec_Alineadas.clustal
```

4.5 Lectura de Secuencias Alineadas

Para poder hacer uso de las secuencias alineadas tenemos que almacenarla en una variable o tambien llamado objeto alineamiento. El siguiente algoritmo recibe la direccion del archivo clustal y retorna el objeto alignment.

```
In [5]: 1 from Bio import AlignIO
2
3 def leer_SecAli(input_clustal = 'data_gen/Sec_Alineadas.clustal'):
4
5     """ Recibe la direccion del archivo clustal con la secuencias aliendas
6     Retorna un objeto alineamiento"""
7
8     aln = open(input_clustal, "r")
9     # AlignIO para leer el archivo de alineamiento en formato 'clustal' format
10    alignment = AlignIO.read(aln, "clustal")
11    return alignment
12
In [6]: 1 aln = leer_SecAli()
2 print("\nLas secuencias alineadas son\n",aln)
```

```
Las secuencias alineadas son
SingleLetterAlphabet() alignment with 4 rows and 347 columns
MNFLSKPIIYSTLIMGLITLFLSSHWLLMWIGLEMSMLAMIPIM...ILF AJE26518.1
MNPLIFTILLMTLILSTMIVITSSHWLFAWIGLEINTMAIIPIM...TLL NP_944712.1
MKPPILIIIMSTVISGTMIVMTASHWLMWIGFEMNLLAIIPIL...ILD AIY56286.1
MNPLIFSIILLTVMAGTLIVMISSHWLFIWIGFEMNMLAIIPIL...VLY ACJ45786.1
```

4.6 Creación de la Matriz de Distancias

Usando el parsing del alienamiento podemos obtener la distancia (o diferencia) entre todas las secuencias.

Esto nos indica, para cada par de secuencias cuan diferentes son. Para el caso ejemplo usamos como parametro 'identity', tambien esta el parametro 'blosum62' que toma diferentes valores de transición y transversión.

```
In [9]: 1 from Bio.Phylo.TreeConstruction import DistanceCalculator # crear la matriz de distancias
2 from Bio.Phylo.TreeConstruction import DistanceTreeConstructor
3
4 calculator = DistanceCalculator('identity')
5 # añade la matriz de distancias al objeto calculator y lo retorna
6 dm = calculator.get_distance(aln)
7 print(dm)

AJE26518.1      0
NP_944712.1    0.40634005763688763   0
AIY56286.1     0.40922190201729103   0.3631123919308358   0
ACJ45788.1     0.4005763688760807   0.3025936599423631   0.27953890489913547   0
AJE26518.1      NP_944712.1      AIY56286.1      ACJ45788.1
```

4.7 Creación del Arbol Filogenetico

Finalmente, podemos construir un arbol filogenético a partir de las distancias entre todas las secuencias. Se crea una función donde se pueda escoger el tipo de arbol a crear entre el upgma y nj(union de vecinos).

```
In [10]: 1 from Bio.Phylo.TreeConstruction import DistanceTreeConstructor
2
3 def create_Tree(alignment, tipo = 'upgma'):
4     """Se recibe de entrada el alimenamiento y el tipo de arbol nj o upgma
5     Genera ell arbol filogenetico
6     """
7     print('Creando el arbol filogenetico .....')
8     # 3. Creamos la matriz de distancias
9     calculator = DistanceCalculator('identity')
10    # añade la matriz de distancias al objeto calculator y lo retorna
11    dm = calculator.get_distance(alignment)
12
13    #initialize a DistanceTreeConstructor object based on our distance calculator object
14    constructor = DistanceTreeConstructor(calculator)
15
16    #build the tree
17    if tipo == 'upgma':
18        tree = constructor.upgma(dm)
19    if tipo == 'nj':
20        tree = constructor.nj(dm)
21
22    return tree
```

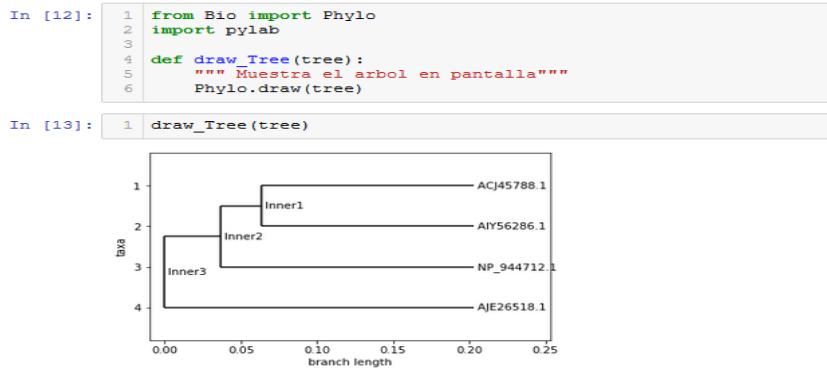
```
In [11]: 1 tree = create_Tree(aln)
2 print(tree)

Creando el arbol filogenetico .....
Tree(rootted=True)
  Clade(branch_length=0, name='Inner3')
    Clade(branch_length=0.03638328530259366, name='Inner2')
      Clade(branch_length=0.026657060518731984, name='Inner1')
        Clade(branch_length=0.13976945244956773, name='ACJ45788.1')
        Clade(branch_length=0.13976945244956773, name='AIY56286.1')
      Clade(branch_length=0.16642651296829972, name='NP_944712.1')
    Clade(branch_length=0.20280979827089338, name='AJE26518.1')
```

Una vez creado el objeto arbol pasamos mostrar el arbol

Mostrar Arbol

Para mostrar el arbol usamos la funcion **Phylo.draw()**, a partir de este arbol se puede realizar el análisis.



4.8 Interfaz Grafica

Todos estos algoritmos y otros se encuentran como funciones en una libreria llamada **BioTool**, el cual es usado en la implementacion de la interfaz grafica. Para la interfaz grafica se utiliza las siguientes librerias, estas ya vienen por defecto con python.

- from tkinter import *
- from tkinter import messagebox
- from tkinter.ttk import *
- from PIL import Image, ImageTk

No se muestra el codigo porque posee demasiadas lineas. La intefaz final se muestra en la siguiente imagen.

5 Resultados

Una descripción de los resultados [esperados en el caso de la propuesta]. Un reporte integrando los resultados proporcionados por la herramienta

5.1 Verificar la fiabilidad del árbol construido

Para la fase beta se encontro el siguiente arbol.

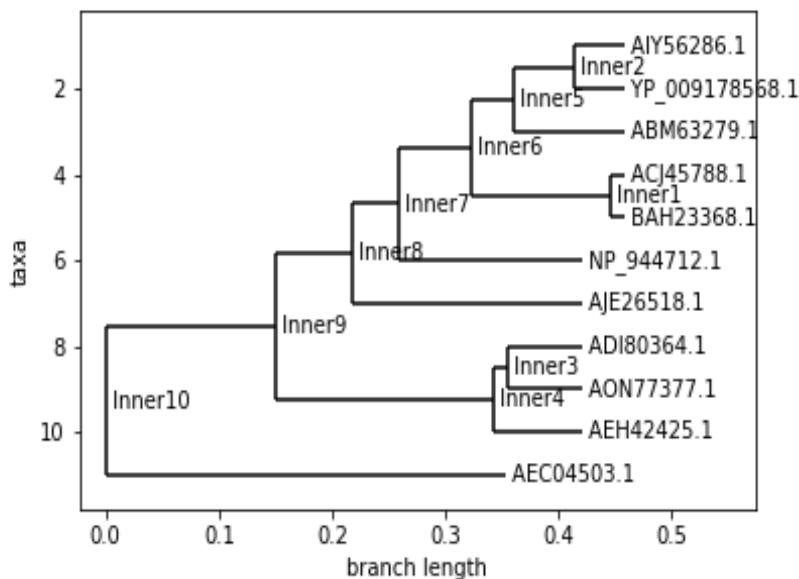


Fig. Arbol utilizando el metodo UPGMA

5.2 Analizar el árbol filogenético

Apartir del arbol filogenetico se podra descubrir/Mostrar/Analizar las relaciones evoluticas de las especies endemicas escogidas.

5.2.1 Modelamiento de la Estructura de Proteinas

En el analisis se encuentra el modelamiento de la Estructura de Proteinas

6 Conclusiones

Incluye las ventajas y desventajas del enfoque utilizado, aspectos inesperados del proyecto, trabajo futuro, etc.

7 Apéndice