

Algoritmo Smith-Waterman

- Laura D. Becerra Largo
- Jonnatan N. Hincapié Hernández
- Alejandro Valencia Ossa
- Mateo Cañavera Aluma

Universidad Nacional De Colombia

Contenido



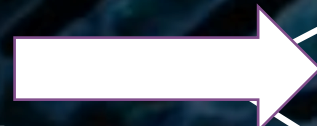
- Contexto
- Problema a resolver
- Implementación del algoritmo en código
 - Lectura de secuencias
 - Matriz de puntuación
 - Matriz de recorrido
 - Alineamiento y escritura del archivo de salida
- Archivo de salida
- Otras implementaciones del algoritmo Smith-Waterman
- Conclusiones

¿Qué hace?

**Alineamiento
local**

$S[1..n]$

$T[1..m]$



A

B



**Subcadenas
mayor
coincidencia**

¿Cómo?

1)

• 1) 1)

1)

$$\begin{aligned} V(i, 0) &= 0 \text{ for } 0 \leq i \leq n \\ V(0, j) &= 0 \text{ for } 0 \leq j \leq m \end{aligned}$$

• 1)

2)

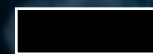
$$V(i, j) = \max \begin{cases} 0 \\ V(i-1, j-1) + \delta(S[i], T[j]) \\ V(i-1, j) + \delta(S[i], -) \\ V(i, j-1) + \delta(-, T[j]) \end{cases}$$

Ejemplo:

$S = ACAATCG$

$T = CTCATGC$

	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0	0	0	0	2	1	0	0
C	0	2	1	2	1	1	0	2
A	0	1	1	1	4	3	2	1
A	0	0	0	0	3	3	2	1
T	0	0	2	1	2	5	4	3
C	0	2	1	4	3	4	4	6
G	0	1	1	3	3	3	6	5



CAATCG



C-AT-G

Implementación del algoritmo en código



Leer secuencias

Matriz de puntuación

Matriz de recorrido

Alineamiento y resultados

Llamar funciones anteriores

Leer Secuencias

```
#Se importa libreria Biopython, que permite leer secuencias geneticas en formato fasta
""" Este Fracmento de codigo se ejecuta solo si es en google colaboratory
try:
    import google.colab
    !pip install biopython
except ImportError:
    pass
"""

#Se importa las librerias numpy y pandas para operar matrices que se obtendran en el algoritmo, además de las funciones necesarias de Biopython para leer el archivo tipo
import numpy as np
import pandas as pd
from Bio import SeqIO
import math
import sys

#Se lee el archivo que contiene las secuencias a analizar y se guardan en la lista 'record'
#Este algoritmo compara la primera secuencia del archivo con la demás
record = list(SeqIO.parse(sys.argv[1], "fasta"))
"""Solo se ejecuta si es en google colaboratory, eliminando la linea anterior a este mensaje
record = list(SeqIO.parse("/content/prueba4.fasta", "fasta"))"""
```


Matriz de Puntuación

```
def matriz_puntuacion(record):
    ##Guarda la matriz de puntuación de la secuencia con mayor similitud
    matrix_max = 0
    ##Guarda en un diccionario el mayor coeficiente obtenido en la matriz de puntuación para cada secuencia comparada
    coef_simil={}
    ##Guarda la posición del mayor coeficiente en la matriz de puntuación de la secuencia con mayor similitud
    coef_max_posicion = [0,0]
    ##Guarda el mayor coeficiente obtenido en la matriz de puntuación de la secuencia con mayor similitud
    coef_max = 0
    ##Guarda posición de la secuencia en la lista de secuencias 'record'
    sec_pos = 0

    #para acceder a cada secuencia se hace por medio de un ciclo:
    n = 0
    for r in range(len(record)-1):
        ##Guarda longitud de la secuencia con la que se comparan las demás
        fila = len(record[0]) + 1
        ##Guarda longitud de la secuencia que se comparará con la primera
        columna = len(record[r+1]) + 1
        ##se crea matriz de puntuación con todas sus entradas en cero
        matrix = np.zeros(shape=(fila, columna), dtype=int)
        ##Guarda la posición del mayor coeficiente en la matriz de puntuación
        max_posicion = [0,0]
        ##Guarda el mayor coeficiente obtenido en la matriz de puntuación
        val_max = 0

        #Este ciclo anidado se encarga de comparar la primera secuencia con la 'r' secuencia por medio del Algoritmo de Smith-
        ##i representará el i-ésimo nucleotido de la primera secuencia de ARN
        for i in range(1,fila):
            ##j representará el j-ésimo nucleotido de la 'r' secuencia de ARN a comparar
            for j in range(1,columna):
                ##Se compara el nucleotido i con cada nucleotido j de la 'r' secuencia y se le asigna una puntuación de acuerdo
                if record[0][i-1] == record[r+1][j-1]:##buscar otro metodo(numpy)
                    coincidencia = 2
                else:
                    coincidencia = -1
                ##se calcula la relación de recurrencia que indica el algoritmo de Smith-Waterman
                diagonal=matrix[i-1][j-1] + coincidencia
                arriba = matrix[i-1][j] + (-1)
                izquierda = matrix[i][j-1] + (-1)
                ##se elige la puntuación de similitud con la relación de recurrencia y se guarda en la matriz creada para tal fin (matrix)
                matrix[i][j] = max(diagonal, arriba, izquierda,0)
                ##Se guarda el mayor coeficiente de puntuación y su posición en la matriz de puntuación
                if val_max <= matrix[i][j]:
                    val_max = matrix[i][j]
                    max_posicion[0]=i
                    max_posicion[1]=j
            n +=1
        ##fin del ciclo anidado##
        ##Se guarda el mayor coeficiente de puntuación para cada secuencia en un diccionario, donde el key es la identificación de la secuencia
        id = record[r+1].id
        coef_simil.setdefault(id,val_max)
        ##Se guarda el con mayor coeficiente de puntuación entre todas las secuencias con su respectiva matriz y posición del coeficiente en esta ultima
        if coef_max <= val_max:
            coef_max = val_max
            coef_max_posicion = max_posicion
            matrix_max = matrix
            sec_pos = n
        ##fin del ciclo de comparación##
    return [coef_max_posicion, sec_pos, matrix_max, coef_simil]
```

	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0	0	0	0	2	1	0	0
C	0	2	1	2	1	1	0	2
A	0	1	1	1	4	3	2	1
A	0	0	0	0	3	3	2	1
T	0	0	2	1	2	5	4	3
C	0	2	1	4	3	4	4	6
G	0	1	1	3	3	3	6	5

```
##Se calcula la relación de recurrencia que indica el algoritmo de Smith-Waterman
diagonal=matrix[i-1][j-1] + coincidencia
arriba = matrix[i-1][j] + (-1)
izquierda = matrix[i][j-1] + (-1)
##se elige la puntuación de similitud con la relación de recurrencia y se guarda en la matriz creada para tal fin (matrix)
matrix[i][j] = max(diagonal, arriba, izquierda,0)
##Se guarda el mayor coeficiente de puntuación y su posición en la matriz de puntuación
if val_max <= matrix[i][j]:
    val_max = matrix[i][j]
    max_posicion[0]=i
    max_posicion[1]=j
n +=1
##fin del ciclo anidado##
##Se guarda el mayor coeficiente de puntuación para cada secuencia en un diccionario, donde el key es la identificación de la secuencia
id = record[r+1].id
coef_simil.setdefault(id,val_max)
##Se guarda el con mayor coeficiente de puntuación entre todas las secuencias con su respectiva matriz y posición del coeficiente en esta ultima
if coef_max <= val_max:
    coef_max = val_max
    coef_max_posicion = max_posicion
    matrix_max = matrix
    sec_pos = n
    ##fin del ciclo de comparación##
return [coef_max_posicion, sec_pos, matrix_max, coef_simil]
```


Matriz de recorrido

```
def matriz_recorrido(record, matrix_max, sec_pos):
    ##se crea matriz que indica el camino de similitud para la secuencia con mayor similitud respecto a la primera secuencia
    matrix2ruta = np.zeros(shape=(len(matrix_max),len(matrix_max[0])), dtype=int)
    ##se recorre la matriz de maxima similitud siguiendo los parametros descritos en el algoritmo de Smith-Waterman para crear
    # la nueva matriz que indica el recorrido
    for i in range(1,len(matrix_max)):
        for j in range(1,len(matrix_max[0])):
            ##se calcula nuevamente la puntuación de coincidencia (solo para la matriz de maxima similitud)
            if record[0][i-1] == record[sec_pos][j-1]:
                coincidencia = 2
            else:
                coincidencia = -1
            ##se calcula la relación de recurrencia que indica el algoritmo de Smith-Waterman
            diagonal=matrix_max[i-1][j-1] + coincidencia
            arriba = matrix_max[i-1][j] + (-1)
            izquierda = matrix_max[i][j-1] + (-1)
            ##se llena la matriz de recorrido
            if matrix_max[i][j] == 0 :
                matrix2ruta[i][j]=0
            elif matrix_max[i][j] == diagonal:
                matrix2ruta[i][j]=1
            elif matrix_max[i][j] == arriba:
                matrix2ruta[i][j]=2
            elif matrix_max[i][j] == izquierda:
                matrix2ruta[i][j]=3
    ###fin del ciclo para llenado de matriz de recorrido###
    return matrix2ruta
```

	-	A	G	C	A	T	G	C
-	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

Alineamiento y resultados

```
def resultados(record, matrix2ruta, coef_max_posicion, coef_simil, sec_pos):
    ##se 'renombrar' la lista que contiene la posición del mayor coeficiente de puntuación
    max_posicion1= coef_max_posicion.copy()
    ##se crea el string que guardará la secuencia principal de comparación (al revés, debido
    #a que, por el algoritmo, el camino siempre compara de atras hacia adelante las secuencias)
    principal=""
    ##se crea el string que guardará la secuencia que fue comparada con la principal y tuvo
    #mayor similitud (al revés, debido a que, por el algoritmo, el camino siempre compara
    #521{de atras hacia adelante las secuencias)
    comparacion=""
    ##se llenan los strings de acuerdo a la matriz de recorrido ordenandolos para que queden alineados
    while matrix2ruta[max_posicion1[0]][max_posicion1[1]] != 0:
        if matrix2ruta[max_posicion1[0]][max_posicion1[1]]== 1:
            principal=principal+record[0][max_posicion1[0]-1]
            comparacion=comparacion+record[sec_pos][max_posicion1[1]-1]
            max_posicion1[0]=max_posicion1[0]-1
            max_posicion1[1]=max_posicion1[1]-1

        elif matrix2ruta[max_posicion1[0]][max_posicion1[1]]== 2:
            principal=principal + record[0][max_posicion1[0]-1]
            comparacion=comparacion + '-'
            max_posicion1[0]=max_posicion1[0]-1
            max_posicion1[1]=max_posicion1[1]

        elif matrix2ruta[max_posicion1[0]][max_posicion1[1]]== 3:
            principal=principal + '-'
            comparacion=comparacion + record[sec_pos][max_posicion1[1]-1]
            max_posicion1[0]=max_posicion1[0]
            max_posicion1[1]=max_posicion1[1]-1

    ##se crea el string que guardará los simbolos correspondiente al alineamiento entre las secuencias
```

Alineamiento y resultados

```
##se crea el string que guardará los símbolos correspondiente al alineamiento entre las secuencias
alin=""
##Se crea contador que permite saber el total de nucleotidos que se alinean entre ambas secuencias
favor=0
##Se invierte los strings que indican las secuencias comparadas (al revés)
comparacion=comparacion[::-1]
principal=principal[::-1]

##se llena la linea que indica el alineamiento entre secuencias comparando las secuencias por medio de un ciclo
for i in range(len(comparacion)):
    if principal[i]==comparacion[i]:
        ##'|'indica que en ambas cadenas de ARN se encuentra el mismo nucleotido en la misma posición
        alin=alin+"|"
        favor = favor + 1
    else:
        ##' ' indica que no hay similitud en la i-ésima posición de las secuencias
        alin=alin+" "

##se convierte el diccionario con los coeficientes de similitud de cada secuencia en un DataFrame
dataf = pd.DataFrame([[key, coef_simil[key]] for key in coef_simil.keys()], columns=['Id de la secuencia', 'Coef. de similitud'])
##Se calcula el porcentaje de similitus entre las secuencias
P = (favor/len(record[0]))*100
per = round(P,3)
###dividir el alineamiento de la secuencia#####
M=math.ceil(len(principal)/100)
##se crea y se escribe en un archivo de texto toda la información analizada en el algoritmo
doc = open('comparacion_similitud.txt','w')
doc.write(f'La secuencia principal, con la que se compararon las demás: {record[0].id}')
doc.write(f'\n')
doc.write(f'La secuencia con mayor similitud a la secuencia principal fue {record[sec_pos].id} con un')
doc.write(f'\n')
```

Alineamiento y resultados

```
doc.write(f'porcentaje de similitud del {per}%')
doc.write(f'\n')
doc.write(f'\n')
doc.write(f'A continuación se muestra el alineamiento entre ambas secuencias:')
doc.write(f'\n')
for i in range(M):
    doc.write(f'{principal[100*i:100*(i+1)]}')
    if i == M-1:
        doc.write(f'----{record[0].id}')
        doc.write(f'\n')
        doc.write(f'{alin[100*i:100*(i+1)]}')
        doc.write(f'\n')
        doc.write(f'{comparacion[100*i:100*(i+1)]}')
        if i == M-1:
            doc.write(f'----{record[sec_pos].id}')
            doc.write(f'\n')
doc.write(f'\n')
doc.write(f'\n')
doc.write(f'Además se presenta la siguiente tabla que relaciona cada una de las secuencias comparadas con su')
doc.write(f'\n')
doc.write(f'coeficiente de similitud, que corresponde a la puntuación maxima de similitud que tuvo cada secuencia')
doc.write(f'\n')
doc.write(f'respecto a la secuencia principal')
doc.write(f'\n')
doc.write(f'\n')
doc.write(f'{dataf}')
doc.close()
```


Llamar funciones anteriores

```
punt = matriz_puntuacion(record)
matr = matriz_recorrido(record, punt[2], punt[1])
resultados(record, matr, punt[0], punt[3], punt[1])
```

Archivo de salida

La secuencia principal, con la que se compararon las demás: gi|645322056|ref|NR_118889.1|
La secuencia con mayor similitud a la secuencia principal fue gi|645322058|ref|NR_118890.1| con un porcentaje de similitud del 93.538%

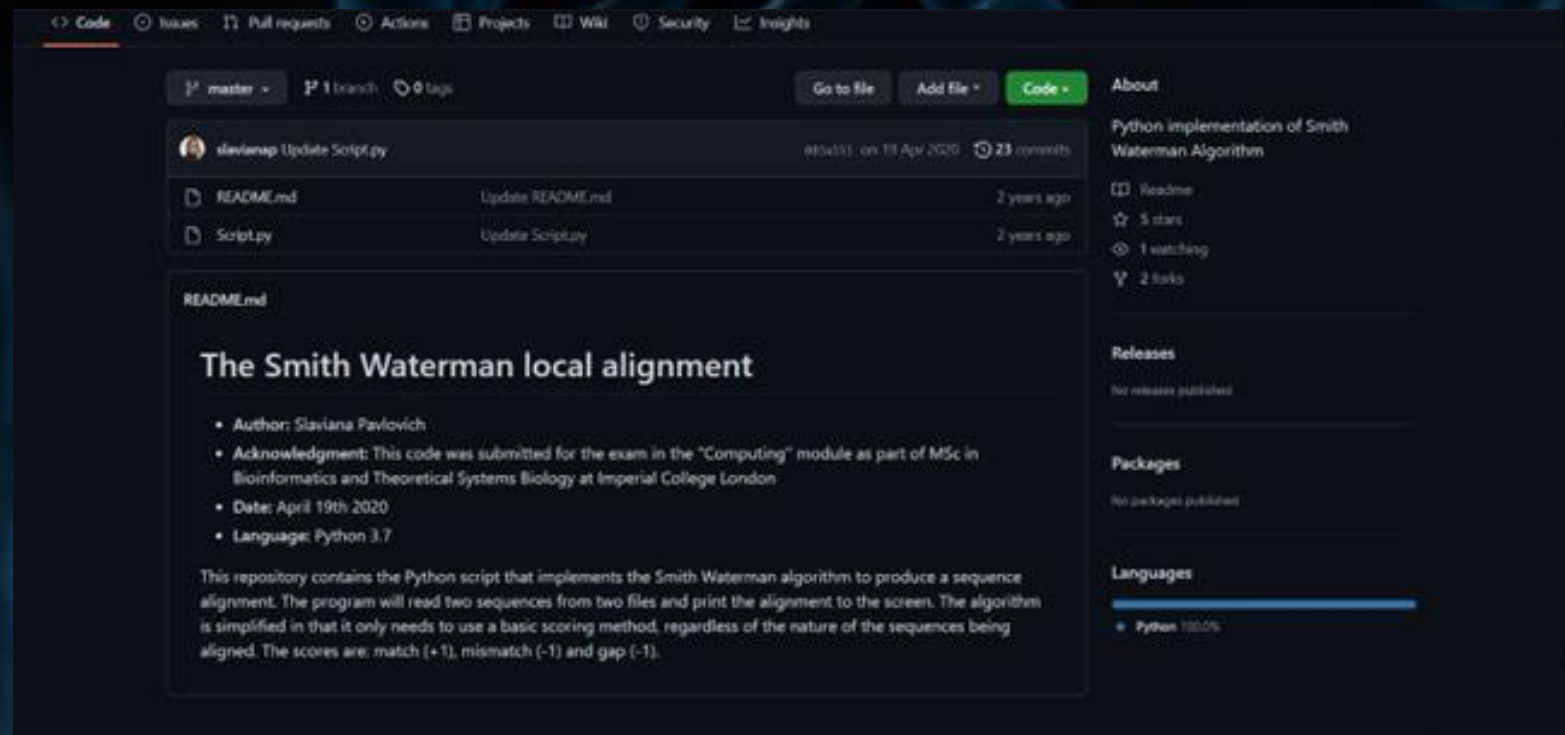
A continuación se muestra el alineamiento entre ambas secuencias:

```
GGTCTNATACCGGATATAACAACATCATGGCATGGTTGGTAGTGGAAAGCTCCGGCGGTACGGGATGAGCCCGCGGCCTATCAGCTTGTTGGTGGGGTAAT
|||||
GGTCTNATACCGGATATGACAACATGATGGCATGGTTGGTTGTGGAAAGCTCCGGCGGTCCGGGATGAGCCCGCGGCCTATCAGCTTGTTGGTGGGGTAAT
GTNAATACGTTCCCGGGCCTTGTACACACCGGNCGTTACGGCATGAAAGNCGGTAACACCCGAA-CCCATGGCCCAACCCGTAA-GGGGGGAGTNGTCGA
|||||
GTNAATACGTTCCCGGGCCTTGTACACACCGGNCGTTACGGCATGAAAGTCGGTAACACCCGAAGCCCATGGCCCAACCCGTAAAGGGGGGAGTGGTCGA
AGGTGGGACTNGCGAT----gi|645322056|ref|NR_118889.1|
|||||
AGGTGGGACTGGCGAT----gi|645322058|ref|NR_118890.1|
```

Además se presenta la siguiente tabla que relaciona cada una de las secuencias comparadas con su coeficiente de similitud, que corresponde a la puntuación máxima de similitud que tuvo cada secuencia respecto a la secuencia principal

	Id de la secuencia	Coef. de similitud
0	gi 645322068 ref NR_118899.1	2051
1	gi 444303911 ref NR_074334.1	1415
2	gi 645322058 ref NR_118890.1	2332

Otras implementaciones del Algoritmo Smith-Waterman



The screenshot shows a GitHub repository page for a project named "slaviana Update Script.py". The repository is located at "0054551" and was created on "19 Apr 2020". It has "23 commits". The repository contains two files: "README.md" and "Script.py", both updated "2 years ago". The "README.md" file is open, showing the title "The Smith Waterman local alignment". The README text includes the author's name, a submission acknowledgment, the date, and the language. It also describes the repository's purpose and the scoring method used. The right sidebar shows the repository's statistics: "About" (Python implementation of Smith Waterman Algorithm), "Readme" (5 stars, 1 watching, 2 forks), "Releases" (No releases published), "Packages" (No packages published), and "Languages" (Python 100.0%).

Code Issues Pull requests Actions Projects Wiki Security Insights

master P 1 branch 0 tags

Go to file Add file Code

slaviana Update Script.py 0054551 on 19 Apr 2020 23 commits

README.md	Update README.md	2 years ago
Script.py	Update Script.py	2 years ago

README.md

The Smith Waterman local alignment

- Author: Slaviana Pavlovich
- Acknowledgment: This code was submitted for the exam in the "Computing" module as part of MSc in Bioinformatics and Theoretical Systems Biology at Imperial College London
- Date: April 19th 2020
- Language: Python 3.7

This repository contains the Python script that implements the Smith Waterman algorithm to produce a sequence alignment. The program will read two sequences from two files and print the alignment to the screen. The algorithm is simplified in that it only needs to use a basic scoring method, regardless of the nature of the sequences being aligned. The scores are: match (+1), mismatch (-1) and gap (-1).

About
Python implementation of Smith Waterman Algorithm

Readme
5 stars
1 watching
2 forks

Releases
No releases published

Packages
No packages published

Languages
Python 100.0%

```

def smith_waterman(seq1, seq2):
    # Generating the empty matrices for storing scores and tracing
    row = len(seq1) + 1
    col = len(seq2) + 1
    matrix = np.zeros(shape=(row, col), dtype=np.int)
    tracing_matrix = np.zeros(shape=(row, col), dtype=np.int)

    # Initialising the variables to find the highest scoring cell
    max_score = -1
    max_index = (-1, -1)

    # Calculating the scores for all cells in the matrix
    for i in range(1, row):
        for j in range(1, col):
            # Calculating the diagonal score (match score)
            match_value = Score.MATCH if seq1[i - 1] == seq2[j - 1] else Score.MISMATCH
            diagonal_score = matrix[i - 1, j - 1] + match_value

            # Calculating the vertical gap score
            vertical_score = matrix[i - 1, j] + Score.GAP

            # Calculating the horizontal gap score
            horizontal_score = matrix[i, j - 1] + Score.GAP

            # Taking the highest score
            matrix[i, j] = max(0, diagonal_score, vertical_score, horizontal_score)

            # Tracking where the cell's value is coming from

```

```

        # Tracking where the cell's value is coming from
        if matrix[i, j] == 0:
            tracing_matrix[i, j] = Trace.STOP

```

```

        elif matrix[i, j] == horizontal_score:
            tracing_matrix[i, j] = Trace.LEFT

```

```

        elif matrix[i, j] == vertical_score:
            tracing_matrix[i, j] = Trace.UP

```

```

        elif matrix[i, j] == diagonal_score:
            tracing_matrix[i, j] = Trace.DIAGONAL

```

```

    # Tracking the cell with the maximum score
    if matrix[i, j] >= max_score:
        max_index = (i, j)
        max_score = matrix[i, j]

```

```

    # Initialising the variables for tracing
    aligned_seq1 = ""
    aligned_seq2 = ""
    current_aligned_seq1 = ""
    current_aligned_seq2 = ""
    (max_i, max_j) = max_index

```



```
# Tracing and computing the pathway with the local alignment
while tracing_matrix[max_i, max_j] != Trace.STOP:
    if tracing_matrix[max_i, max_j] == Trace.DIAGONAL:
        current_aligned_seq1 = seq1[max_i - 1]
        current_aligned_seq2 = seq2[max_j - 1]
        max_i = max_i - 1
        max_j = max_j - 1

    elif tracing_matrix[max_i, max_j] == Trace.UP:
        current_aligned_seq1 = seq1[max_i - 1]
        current_aligned_seq2 = '-'
        max_i = max_i - 1

    elif tracing_matrix[max_i, max_j] == Trace.LEFT:
        current_aligned_seq1 = '-'
        current_aligned_seq2 = seq2[max_j - 1]
        max_j = max_j - 1

    aligned_seq1 = aligned_seq1 + current_aligned_seq1
    aligned_seq2 = aligned_seq2 + current_aligned_seq2

# Reversing the order of the sequences
aligned_seq1 = aligned_seq1[::-1]
aligned_seq2 = aligned_seq2[::-1]

return aligned_seq1, aligned_seq2
```

Tiempos promedio de ejecución

- Usando la librería “time” de Python podemos estimar el tiempo de ejecución de ambos algoritmos.

Nuestro Algoritmo

Algoritmo alternativo

- Tiempo promedio por

Tiempo promedio por secuencia :24.68 segundos

secuencia:16.19 segundos

- Tiempo total para 100 secuencias

Tiempo total para 100 secuencias: 41 minutos y 7.92 segundos

26 minutos y 58.96 segundos

National Center of Biotechnology Information



	Description	Scientific Name	Max Score	Total Score	Query Cover	E value	Per. Ident	Acc. Len	Accession
✓	gi 645322058 ref NR_118890.1 Actinokineospora fastidiosa strain ATCC 31181 16S ribosomal RNA gene..partial sequence		1865	1865	100%	0.0	92.28%	1331	Query_43611
✓	gi 645322079 ref NR_118908.1 Amycolatopsis methanolica strain 239 16S ribosomal RNA gene..partial sequence		1829	1829	97%	0.0	91.95%	1343	Query_43613
✓	gi 219846494 ref NR_026086.1 Mycobacterium komossense strain Ko2 16S ribosomal RNA gene..partial sequence		1763	1763	100%	0.0	89.82%	1462	Query_43688
✓	gi 265678909 ref NR_029217.1 Mycobacterium aurum strain 358 16S ribosomal RNA gene..partial sequence		1731	1731	99%	0.0	89.59%	1458	Query_43677
✓	gi 645322049 ref NR_118883.1 Mycobacterium fortuitum subsp. fortuitum strain ATCC 6841 16S ribosomal RNA gene..pa...		1724	1724	100%	0.0	89.03%	1450	Query_43683
✓	gi 265678911 ref NR_029219.1 Mycobacterium aichiense strain 49005 16S ribosomal RNA gene..partial sequence		1722	1722	99%	0.0	89.38%	1456	Query_43675
✓	gi 645322086 ref NR_118914.1 Mycobacterium chubuense strain ATCC 27278 16S ribosomal RNA gene..partial sequence		1717	1717	99%	0.0	89.45%	1458	Query_43679
✓	gi 645322087 ref NR_118915.1 Mycobacterium gilvum strain ATCC 43909 16S ribosomal RNA gene..partial sequence		1711	1711	99%	0.0	89.30%	1458	Query_43686
✓	gi 265678910 ref NR_029218.1 Mycobacterium obuense strain 47001 16S ribosomal RNA gene..partial sequence		1708	1708	99%	0.0	89.29%	1458	Query_43691
✓	gi 343206223 ref NR_044815.1 Mycobacterium flavescens strain ATCC 14474 16S ribosomal RNA..partial sequence		1706	1706	100%	0.0	88.78%	1454	Query_43682

1) gi|645322058|ref|NR_118890.1| 2332
2) gi|645322079|ref|NR_118908.1| 2266
3) gi|219846494|ref|NR_026086.1| 2242
4) gi|265678909|ref|NR_029217.1| 2230
5) gi|645322086|ref|NR_118914.1| 2226

6) gi|265678911|ref|NR_029219.1| 2224
7) gi|645322087|ref|NR_118915.1| 2220
8) gi|265678910|ref|NR_029218.1| 2218
9) gi|265678912|ref|NR_029220.1| 2214
10) gi|343206223|ref|NR_044815.1| 2212

gi|645322058|ref|NR_118890.1| Actinokineospora fastidiosa strain ATCC 31181 16S ribosomal RNA gene, partial sequence

Sequence ID: Query_41906 Length: 1331 Number of Matches: 1

Range 1: 27 to 1330 [Graphics](#)

[▼ Next Match](#) [▲ Previous Match](#)

Score	Expect	Identities	Gaps	Strand
1865 bits(2068)	0.0	1207/1308(92%)	12/1308(0%)	Plus/Plus
Query 1	GGTCTNATACCGGATATAACAACATCATGGCATGGTTGGTAGTGGAAAGCTCCGGCGGTAC	60		
Sbjct 27	GGTCTNATACCGGATATGACAACTGATGGCATGGTTGGTGTGGAAAGCTCCGGCGGTCC	86		
Query 61	GGGATGAGCCCGCGGCCTATCAGCTTGTTGGTGGGGTAATGGCCTACCAAGGCGACGACG	120		
Sbjct 87	GGGATGAGCCCGCGGCCTATCAGCTTGTTGGTGGGGTAATGGCCTACCAAGGCGACGACG	146		
Query 121	GGTAGCCGGCCTGAGAGGGTGACCGGCNACACTGGGACTGAGACACGGCCNAGACTCCTA	180		
Sbjct 147	GGTAGCCGGCCTGAGAGGGTGACCGGCCACACTGGGACTGAGACACGGCCNAGACTCCTA	206		
Query 181	CNGGAGGNAGCAGTGGGGAATATTGCNCAATGGGCGAAAGCCTNATGCAGCGACGCCGCG	240		
Sbjct 207	CNGGAGGNAGCAGTGGGGAATNTTGCNCAATGGNNGAAAGCCTNACGCAGCGACGCCGCG	266		
Query 241	TGAGGGATGACGGC-TTCGGGTTGTAAACCTTTTTCGCCAGGGACGAAGCGCAAGTGACG	299		
Sbjct 267	TGGGGGATGACGGCCTTCGGGTTGTAAACCTTTTTCGCCAGGGACGAAGCGCGAGTGACG	326		
Query 300	GTACCTGGAGAAGAAGCACCGGCTAACTACGTGCCAGCAGCCGCGGTAATACGTAGGGTG	359		
Sbjct 327	GTACCTGGAGAAGAAGCGCCGGCTNACTACGTGCCAGCAGCCGCGGTAATACGTAGGGTG	386		
Query 360	CGAGCGTTGTCCGGAATTACCGGGCGTAAAGAGCTNGTAGGCGGTTTGTGCGGTNGTTCG	419		
Sbjct 387	CGAGCGTNGTCCGGAANNATTGGGCGTAAAGATCTNGTAGGCGGTTNGTCTCGTCGGCCG	446		
Query 420	TGAAAACCTCCACG--CTNAACGTTGAGCGTGCGGGCGATACGGGCAGACTNGAGTTCGGT	477		
Sbjct 447	TGAAAAC--CAGGGACTAAACTCTGGGCCTGCGGTGATACGGGCAGACTNGAGTTCGGT	504		

Fast and exact sequence alignment with the Smith–Waterman algorithm: The SwissAlign webserver

Gábor Iván, Dániel Bánky, Vince Grolmusz 

Show more 

+ Add to Mendeley  Share  Cite

The BLAST (Basic Local Alignment Search Tool) algorithm ([Altschul et al., 1990](#)) and its versions are among the most important algorithmic applications in biology. The speed of BLAST made it a much more frequently used algorithm, than the exact Smith–Waterman algorithm for sequence alignments ([Smith and Waterman, 1981](#)). Most of the users of the BLAST algorithm are not aware of the speed they need to pay with accuracy: extensive studies witnessed that the Smith–Waterman algorithm finds alignments that are overlooked by BLAST ([Pearson, 1991](#), [Pearson, 1995](#), [Shpaer et al., 1996](#)).

Several methods were reported to speed up the slow, but exact and reliable Smith–Waterman algorithm ([Smith and Waterman, 1981](#)) with artificial intelligence applications ([Eddy, 2011](#)), with parallelization ([Bandyopadhyay and Mitra, 2009](#), [Hasan et al., 2011](#)), with CPU-specific command-sets ([Rognes and Seeberg, 2000](#), [Farrar, 2007](#), [Rognes, 2011](#), [Szalkowski et al., 2008](#)), clustering ([Itoh et al., 2004](#)), preprocessing ([Itoh et al., 2005](#)), or advanced heuristic approaches ([Smith, 2006](#)), some of these improved the Smith–Waterman implementations to achieve a speed comparable to that of the BLAST variants ([Farrar, 2007](#)). All of these methods are important since they make the exact Smith–Waterman algorithm applicable for large