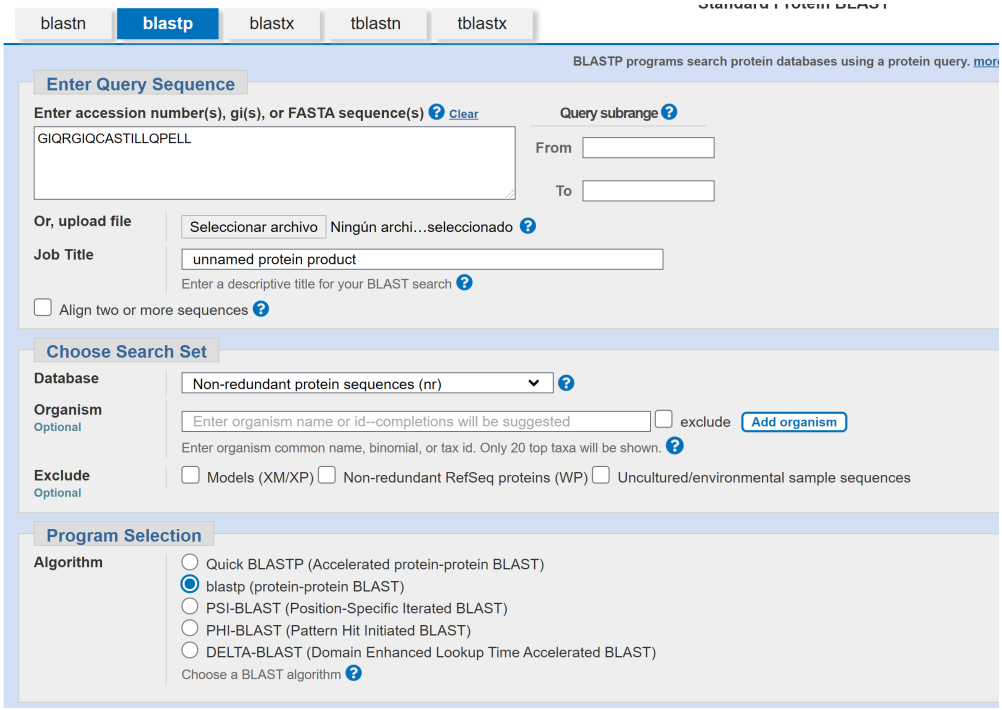
Tarea 1 Bioinformática

December 4, 2021

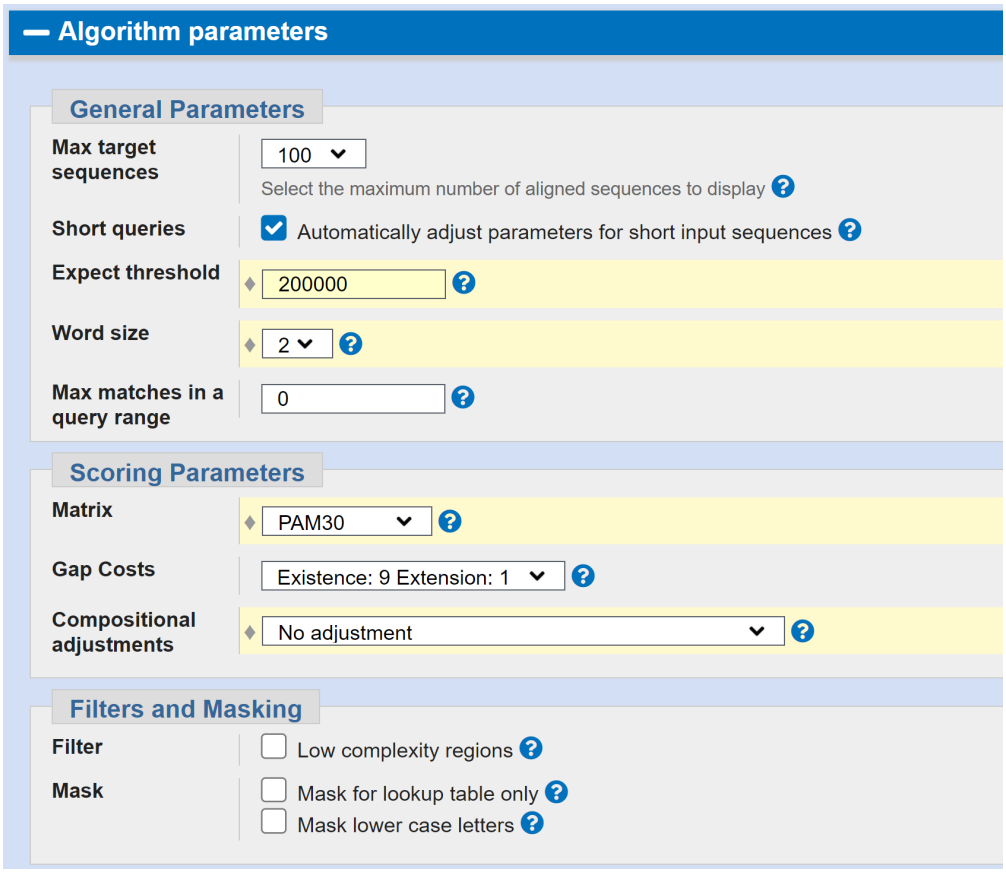
Integrantes:
Juan Pablo Castillo, rol: 201473599-5
Giorgio Pellizzari, rol: 201573534-4

1 Pregunta 1

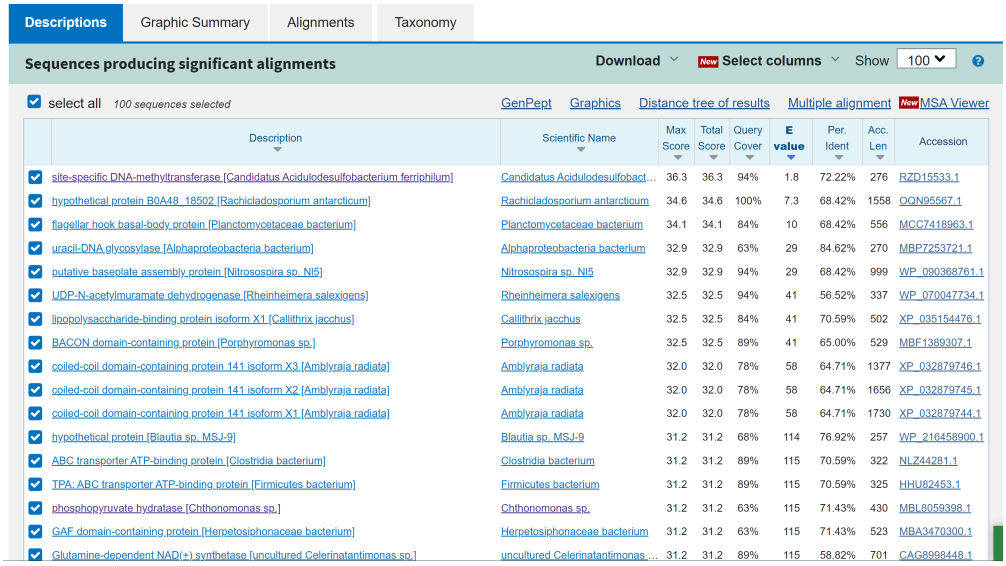
Consulta: **GIQRGIQCASTILLQPELL**



Parámetros:



Resultado: **site-specific DNA-methyltransferase**



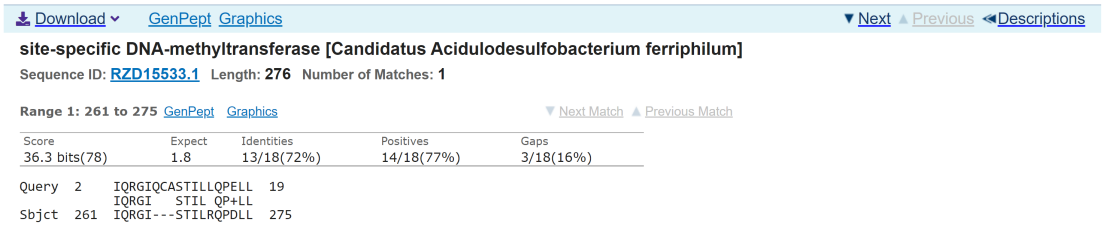
Descriptions | Graphic Summary | Alignments | Taxonomy

Sequences producing significant alignments — Download — New Select columns — Show 100

select all 100 sequences selected — GenPept | Graphics | Distance tree of results | Multiple alignment | New MSA Viewer

Description	Scientific Name	Max Score	Total Score	Query Cover	E value	Per. Ident	Acc. Len	Accession
site-specific DNA-methyltransferase [Candidatus Acidulodesulfobacterium ferriphilum]	Candidatus Acidulodesulfobact…	36.3	36.3	94%	1.8	72.22%	276	RZD15533.1
hypothetical protein B0A48_18502 [Rachicladosporium antarcticum]	Rachicladosporium antarcticum	34.6	34.6	100%	7.3	68.42%	1558	OQN95567.1
flagellar hook basal-body protein [Planctomycetaceae bacterium]	Planctomycetaceae bacterium	34.1	34.1	84%	10	68.42%	556	MCC7418963.1
uracil-DNA glycosylase [Alphaproteobacteria bacterium]	Alphaproteobacteria bacterium	32.9	32.9	63%	29	84.62%	270	MBP7253721.1
putative baseplate assembly protein [Nitrosospira sp. Nl5]	Nitrosospira sp. Nl5	32.9	32.9	94%	29	68.42%	999	WP_090368761.1
UDP-N-acetylmuramate dehydrogenase [Rheinheimera salexigens]	Rheinheimera salexigens	32.5	32.5	94%	41	56.52%	337	WP_070047734.1
lipopolysaccharide-binding protein isoform X1 [Callithrix jacchus]	Callithrix jacchus	32.5	32.5	84%	41	70.59%	502	XP_035154476.1
BACON domain-containing protein [Porphyromonas sp.]	Porphyromonas sp.	32.5	32.5	89%	41	65.00%	529	MBF1389307.1
coiled-coil domain-containing protein 141 isoform X3 [Amblyraja radiata]	Amblyraja radiata	32.0	32.0	78%	58	64.71%	1377	XP_032879746.1
coiled-coil domain-containing protein 141 isoform X2 [Amblyraja radiata]	Amblyraja radiata	32.0	32.0	78%	58	64.71%	1656	XP_032879745.1
coiled-coil domain-containing protein 141 isoform X1 [Amblyraja radiata]	Amblyraja radiata	32.0	32.0	78%	58	64.71%	1730	XP_032879744.1
hypothetical protein [Blautia sp. MSJ-9]	Blautia sp. MSJ-9	31.2	31.2	68%	114	76.92%	257	WP_216458900.1
ABC transporter ATP-binding protein [Clostridia bacterium]	Clostridia bacterium	31.2	31.2	89%	115	70.59%	322	NLZ44281.1
TPA: ABC transporter ATP-binding protein [Firmicutes bacterium]	Firmicutes bacterium	31.2	31.2	89%	115	70.59%	325	HHU82453.1
phosphopyruvate hydratase [Chthonomonas sp.]	Chthonomonas sp.	31.2	31.2	63%	115	71.43%	430	MBL8059398.1
GAF domain-containing protein [Herpetosiphonaceae bacterium]	Herpetosiphonaceae bacterium	31.2	31.2	63%	115	71.43%	523	MBA3470300.1
Glutamine-dependent NAD(+) synthetase [uncultured Celerinatantimonas sp.]	uncultured Celerinatantimonas …	31.2	31.2	89%	115	58.82%	701	CAG8998448.1

Mejor resultado:

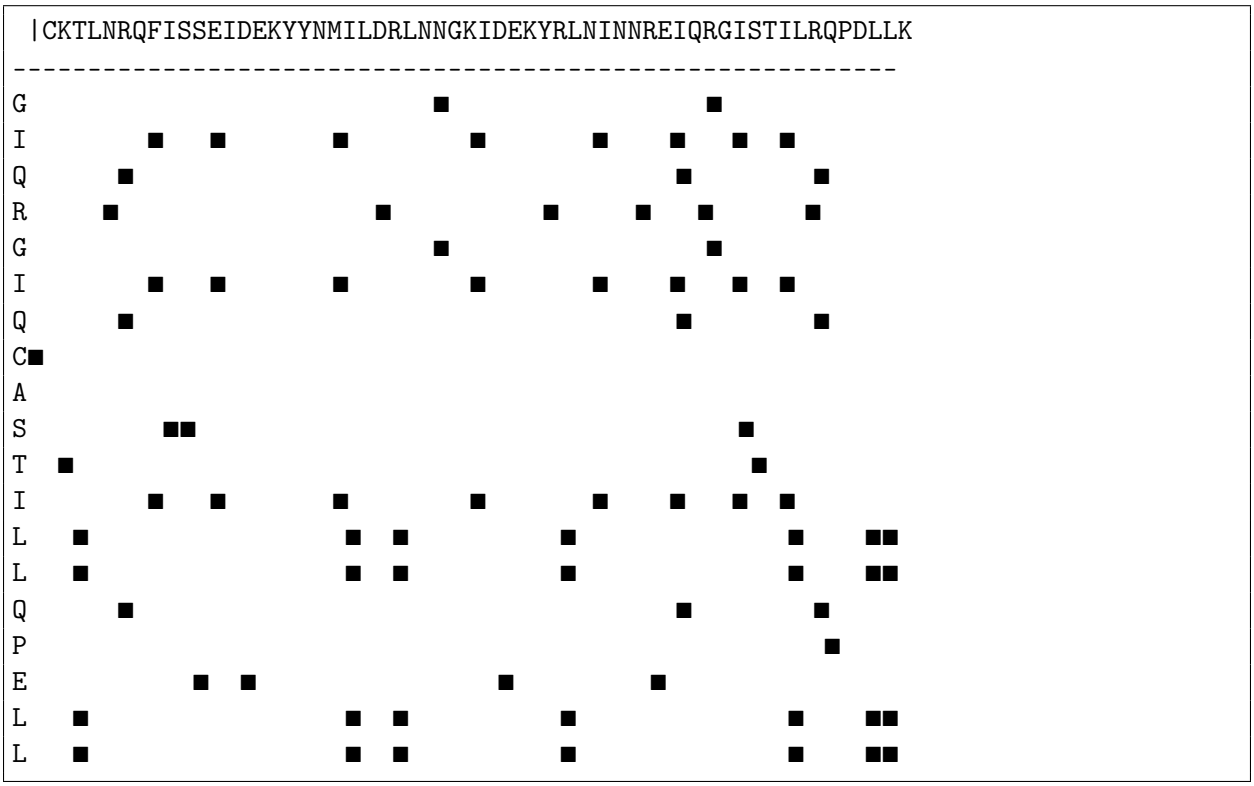


Muestre dos dotplots entre su query (en la forma que se usó en la búsqueda) y un segmento de la proteína encontrada, que incluya el match pero sea más largo que el query (digamos, 3 veces la longitud). El primer dotplot hágalo para las identidades (dotplot simple), y el segundo hágalo indicando dos identidades en ventana de largo 3.

[82]:
```
Query = 'GIQRGIQCASTILLQPELL'
Rsequence = 'CKTLNRQFISSEIDEKYYNMILDRLNNGKIDEKYRLNINNREIQRGISTILRQPDLLK'

def printDotPlot(seq1, seq2, w = 3, s = 3):
    n = len(seq1)
    m = len(seq2)
    M = np.zeros((n, m))

    print(" |", end="")
    for word in seq2:
        print(word, end='')
    print("")
    for _ in range(len(seq2) + 1):
        print("-", end="")
    print("")
    for i in range(n - w + 1):
        print(seq1[i], end="")
        for j in range(m - w + 1):
            count = 0
            for wp in range(w):
                if seq1[i+wp] == seq2[j+wp]:
                    count += 1
            if count >= s:
                M[i, j] = 1
                print(chr(0x25A0), end="")
            else:
                print(" ", end="")
        print("")
```

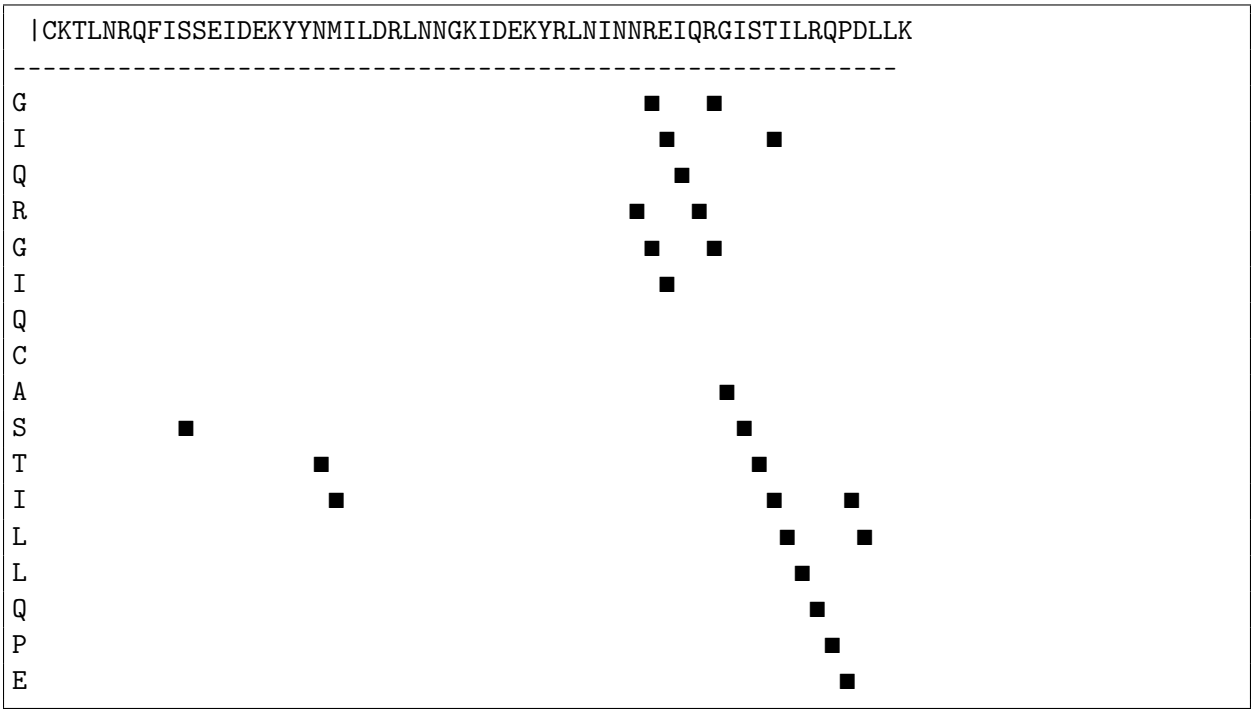
Dot plot identidad simple:

[84]:
```
printDotPlot(Query, Rsequence, w=1, s=1)
```



Dot plot con dos identidades y ventanad de largo 3:

[85]:
```
printDotPlot(Query, Rsequence, w=3, s=2)
```



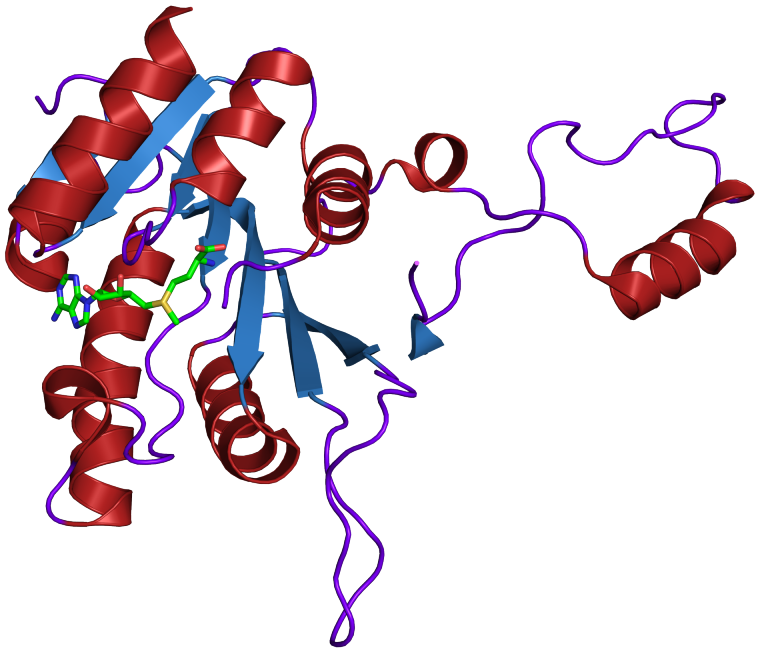
Muestre el alineamiento que dio BLAST entre su nombre y el segmento de la pro-

teína.

```
Query  2    IQRGIQCASTILLQPELL  19
            IQRGI   STIL QP+LL
Sbjct  261  IQRGI---STILRQPDLL  275
```

¿Qué función cumple la proteína? [Para la función: trate de averiguar; por ejemplo, si es una "pitufasa", trate de investigar que significa "pitufar", o qué es la "pitufosis"]. ¿Se sabe con certeza (base experimental), o es una conjetura? ¿Se conoce su estructura 3D? ¿Está etiquetada como miembro de alguna familia de proteínas?

La proteína site-specific DNA methyltransferase (N6_n4_Mtase), cumple la función de catalizar la transferencia de un grupo metilo al ADN. De la proteína se sabe con certeza que existe, y pertenece a la familia de proteinas DNA methylases. Su estructura 3D está dada por la siguiente figura:



El organismo en que la encontró: ¿qué es? (clasificación, descripción, imagen si es que existe [Use The Google!]).

El organismo en que se encontro es Candidatus Acidulodesulfobacterium ferriphilum el cual es una proteobacteria que es muy abundante en ambientes extremadamente ricos en hierro y azufre, de los cuales la fisiologia, biodiversidad y funciones ecologicas se sabe muy poco.

¿Dónde y cómo está codificada la proteína? (Indique la ubicación de la CDS - "coding sequence"- de DNA que la codifica, y acaso está en la hebra primaria o

secundaria, si es continua o tiene interrupciones, si está en cierto cromosoma, si es parte del genoma mitocondrial, etc.; incluya el código de acceso o URL de la secuencia de DNA).

La proteína se encuentra codificada entre las posiciones 240.414 y 241.244 de la hebra primaria, y sin interrupciones en el CDS del organismo.

¿Qué origen tienen los datos? (¿Proyecto de secuenciamiento masivo? ¿Paper específico sobre el gen? ¿Estudio filogenético?)

Los datos fueron publicados por College of Ecology and Evolution, Sun Yat-sen University el 31 de enero de 2019. Y estos no fueron publicados en ningún artículo científico.

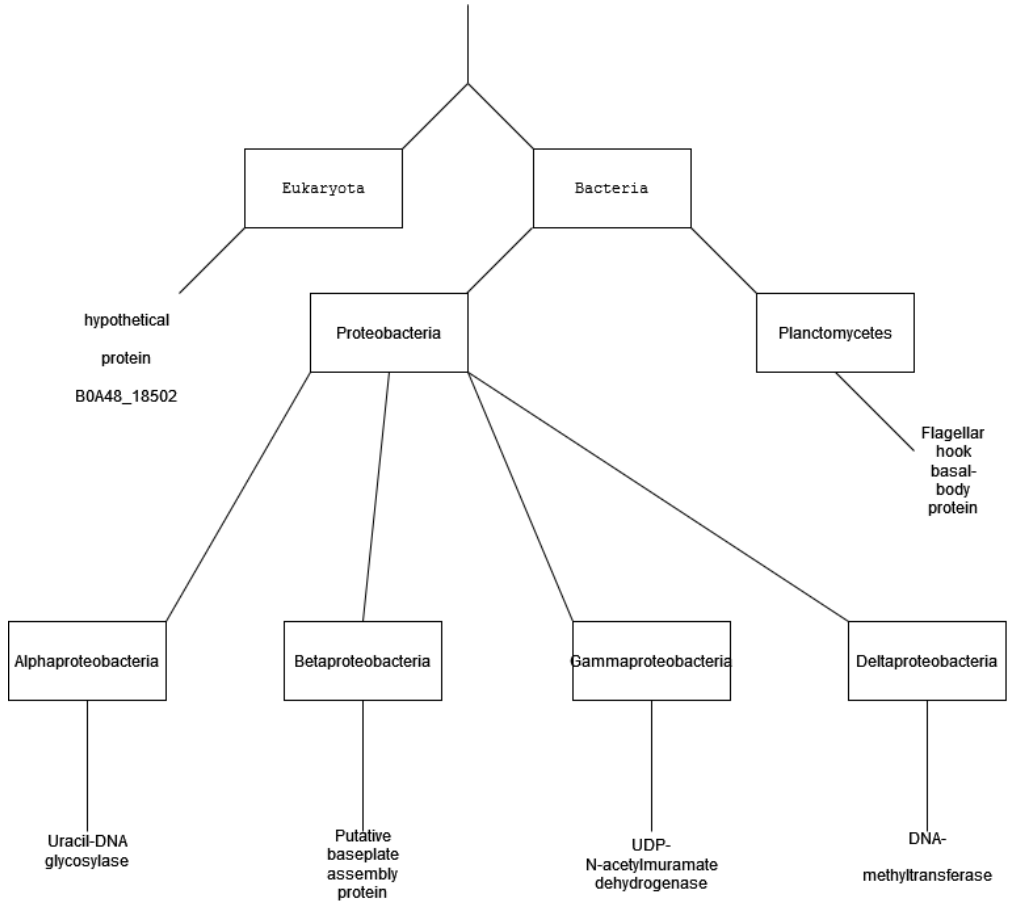
Considerando ahora además los 5 matches siguientes en su búsqueda inicial (por lo tanto, 6 secuencias encontradas) que sean en especies distintas (de modo que si hay más de uno en la misma especie, se ignoran las "repeticiones"),

- **¿En qué organismos están? ¿Son especies cercanas a las del primer match?**

Los 5 matches mas cercanos fueron los siguientes:

- Hypothetical protein B0A48_18502, encontrada en Rachicladosporium antarcticum el cual es un organismo celular muy distinto a nuestro match inicial.
- Flagellar hook basal-body protein, encontrada en Planctomycetaceae bacterium la cual es una bacteria que deriva de los Planctomycetes, no de los Protobacteria como nuestro match, siendo mas cercano que el match anterior.
- Uracil-DNA glycosylase, encontrada en Alphaproteobacteria bacterium al igual que la DNA-methyltransferase es una Protobacteria que luego deriva en la familia Alphaproteobacteria, mientras que la DNA-methyltransferase deriva en las Deltaproteobacteria.
- Putative baseplate assembly protein, encontrada en Nitrosospira sp. Nl5 es una Protobacteria que luego deriva en la familia de las Betaproteobacterias.
- UDP-N-acetylmuramate dehydrogenase, encontrada en Rheinheimera salexigens es una Protobacteria que luego deriva en la familia de las Gammaproteobacterias.

- **Dibuje el "árbol familiar" que muestre las relaciones entre los 6 organismos, de acuerdo a la información en Genbank.**



2 Pregunta 2

[2]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

random.seed(20)

bases = ['A', 'C', 'G', 'T', 'B', 'D']
mutations = ['insert', 'delete', 'replace']
```

[86]:
```
# Función que genera una sequencia aleatoria de largo n
def generateRandomSequence(bases, n = 200):
    sequence = []
    for i in range(n):
        b = random.choice(bases)
        sequence.append(b)

    return sequence
```

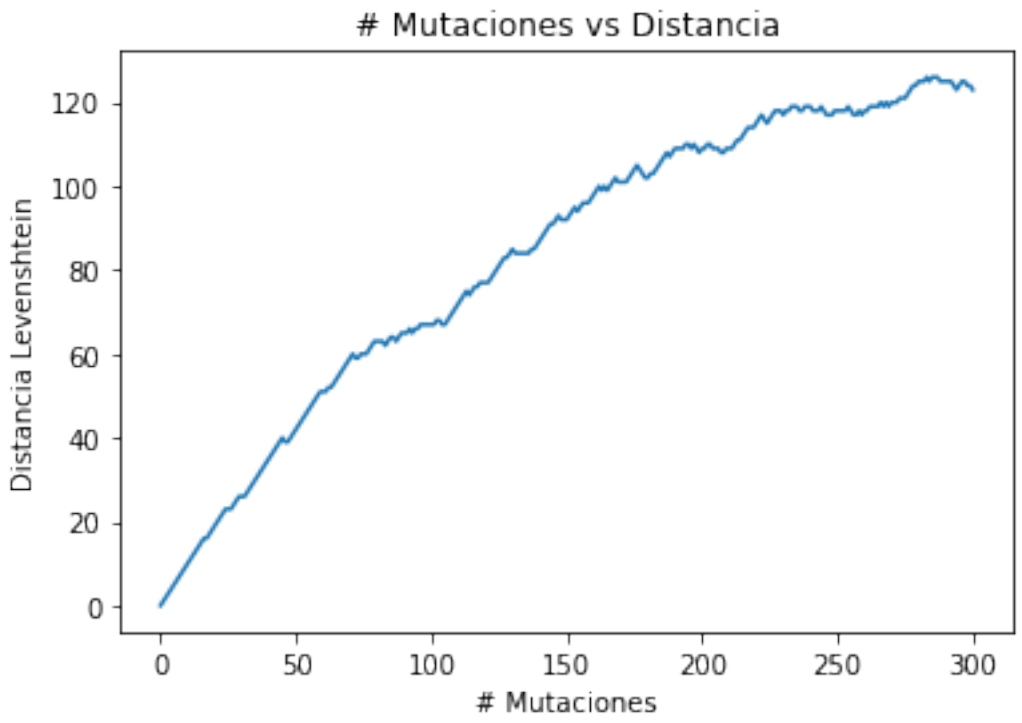
```python
# Función que realiza una mutación aleatoria a la secuencia
def randomMutation(sequence, bases, mutations):
    # Choose random mutation
    m = random.choice(mutations)
    position = random.randint(0, len(sequence)-1)
    aux_sequence = sequence.copy()
    if m == 'insert':
        b = random.choice(bases)
        aux_sequence = []
        j = 0
        for i in range(len(sequence) + 1):
            if i == position:
                aux_sequence.append(b)
            else:
                aux_sequence.append(sequence[j])
                j += 1

    elif m == 'delete':
        del aux_sequence[position]

    elif m == 'replace':
        eq_flag = True
        b = ''
        while eq_flag:
            b = random.choice(bases)
            if b != aux_sequence[position]:
                eq_flag = False

        aux_sequence[position] = b

    return aux_sequence

'''
The distance reflects the total number of single-character
edits required to transform one word into another.
'''
def levenshteinDistance(sequence_a, sequence_b):
    distances = np.zeros((len(sequence_a) + 1, len(sequence_b) + 1))

    for i in range(len(sequence_a) + 1):
        distances[i][0] = i

    for j in range(len(sequence_b) + 1):
        distances[0][j] = j

    a = 0
```

```
    b = 0
    c = 0

    for i in range(1, len(sequence_a) + 1):
        for j in range(1, len(sequence_b) + 1):
            if (sequence_a[i-1] == sequence_b[j-1]):
                distances[i][j] = distances[i - 1][j - 1]
            else:
                a = distances[i][j - 1]
                b = distances[i - 1][j]
                c = distances[i - 1][j - 1]

                if (a <= b and a <= c):
                    distances[i][j] = a + 1
                elif (b <= a and b <= c):
                    distances[i][j] = b + 1
                else:
                    distances[i][j] = c + 1

    return distances[len(sequence_a)][len(sequence_b)]
```

La función para calcular distancia Levenshtein se baso de la siguiente fuente: https://blog.paperspace.com/implementing-levenshtein-distance-word-autocomplete-autocorrect/.

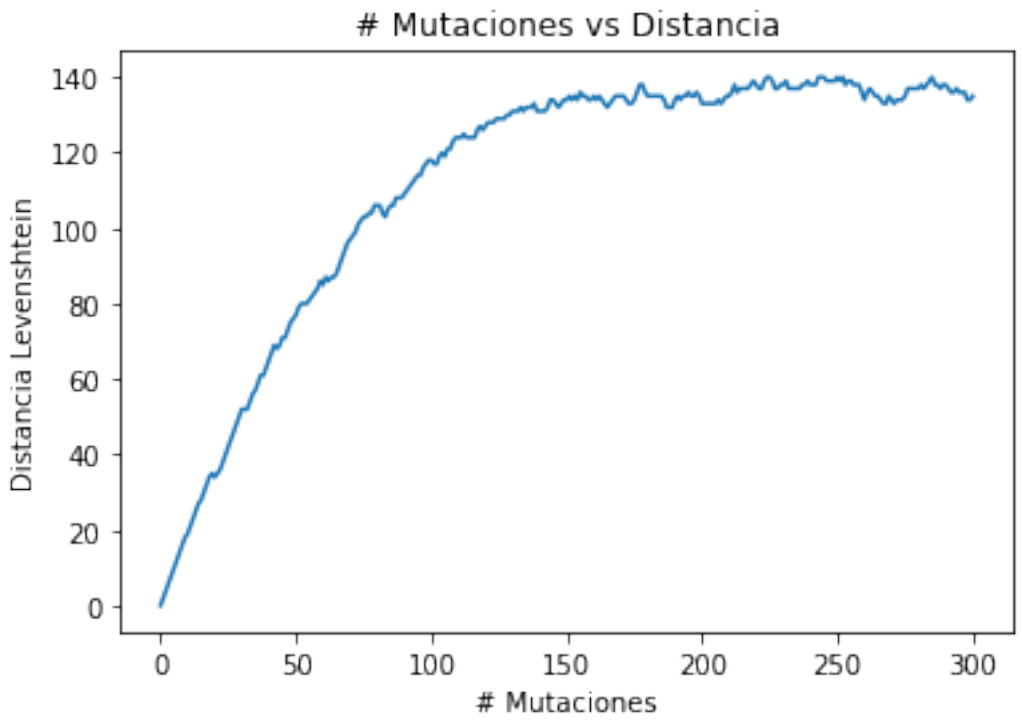
a) **Generar una secuencia, y aplicar M mutaciones; para M entre 0 y 300, grafique la relación entre M y D, donde D es la distancia de Levenshtein entre la secuencia final y la secuencia inicial.**

[4]:
```
M = 300
D = []
sequence_i = generateRandomSequence(bases)
sequence_f = sequence_i.copy()
d = levenshteinDistance(sequence_i, sequence_f)
D.append(d)
for m in range(1, M+1):
    sequence_f = randomMutation(sequence_f, bases, mutations)
    d = levenshteinDistance(sequence_i, sequence_f)
    D.append(d)

plt.plot(range(M+1), D)
plt.title('# Mutaciones vs Distancia')
plt.xlabel('# Mutaciones')
plt.ylabel('Distancia Levenshtein')
plt.show()
```



b) **Genere una secuencia, clónela, y a cada copia aplíquele M mutaciones (de modo que tendrá dos secuencias crecientemente distintas). Grafique la relación entre M y D', donde D' es la distancia entre las dos secuencias que están mutando.**

[5]:
```
Dp = []
# Generate sequences
sequence_1 = generateRandomSequence(bases)
sequence_2 = sequence_1.copy()
d = levenshteinDistance(sequence_1, sequence_2)
Dp.append(d)
for m in range(1, M+1):
    sequence_1 = randomMutation(sequence_1, bases, mutations)
    sequence_2 = randomMutation(sequence_2, bases, mutations)
    d = levenshteinDistance(sequence_1, sequence_2)
    Dp.append(d)

plt.plot(range(M+1), Dp)
plt.title('# Mutaciones vs Distancia')
plt.xlabel('# Mutaciones')
plt.ylabel('Distancia Levenshtein')
plt.show()
```

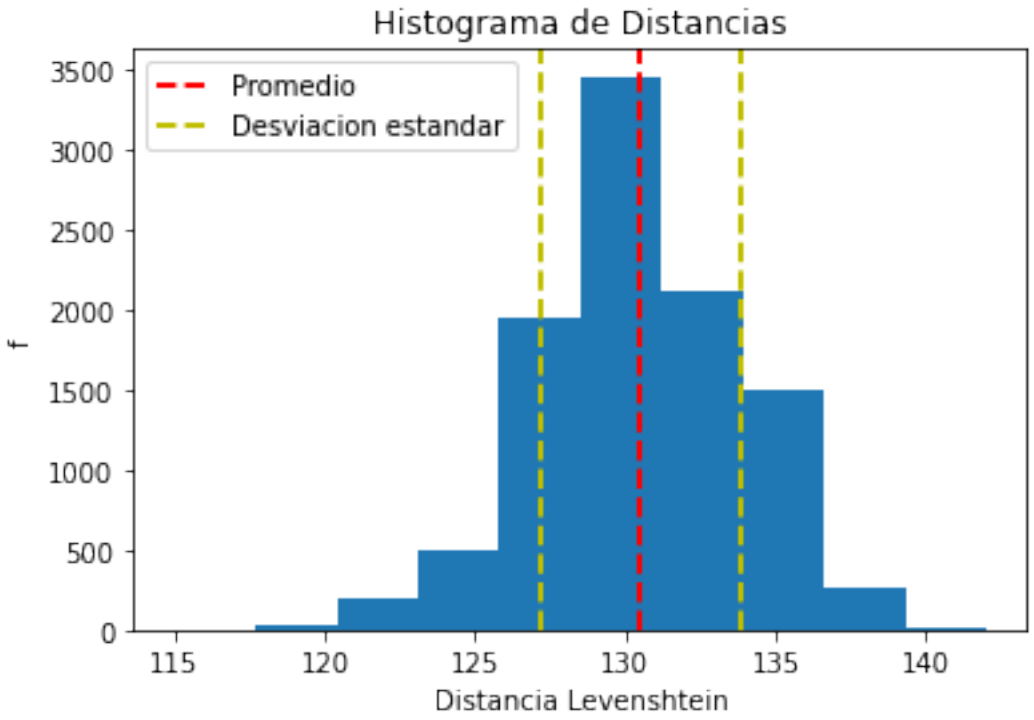


c) **Genere** 10.000 **pares de secuencias (largo** 200 **c/u) y evalúe su distancia de Levenshtein; haga un histograma de la distribución de estos valores, y calcule la media y** σ**.**

[6]:
```
# Generate 10.000 sequences with 200 bases
sequences1 = [generateRandomSequence(bases) for i in range(10000)]
sequences2 = [generateRandomSequence(bases) for i in range(10000)]
distances = []
for i in range(10000):
    s1 = sequences1[i]
    s2 = sequences2[i]
    d = levenshteinDistance(s1, s2)
    distances.append(d)
```

[92]:
```
import statistics

avg = sum(distances)/len(distances)
std = statistics.stdev(distances)

plt.hist(distances)
plt.title('Histograma de Distancias')
plt.xlabel('Distancia Levenshtein')
plt.ylabel('f')
plt.axvline(avg, color='r', linestyle='dashed', linewidth=2)
```

```
plt.axvline(avg-std, color='y', linestyle='dashed', linewidth=2)
plt.axvline(avg+std, color='y', linestyle='dashed', linewidth=2)
plt.legend(["Promedio", "Desviacion estandar"])
plt.show()
print('Promedio: ', avg)
print('Desviacion estandar: ', std)
```




```
Promedio:  130.4721
Desviacion estandar:  3.3291635470478615
```

d) **Considerando (b) y (c), ¿por sobre qué valor de M diría usted que el parentesco entre las secuencias es indetectable (es decir, se entra en la twilight zone)? Tome en cuenta que cuando la distancia ya está dentro de la twilight zone, hace rato que la capacidad de juzgar parentesco ya se perdió.**

Considerando el gráfico anterior, en promedio la distancia entre dos secuencias aleatorias es de aproximadamente 130. Por lo tanto, con una desviación estándar de $3,33$ se puede decir que con una distancia de 133 estamos seguros de que las secuencias no se parecen (no tienen parentesco), debido que la distancia es mayor que si estas dos secuencias fueran escogidas aleatoriamente y calculado su distancia.