Trabajo:

Programa de MSA

Universidad Autónoma de Coahuila

Facultad de Sistemas

Análisis y modelación de sistemas

Profesor Ernesto Ríos Willars

Alumno:

Alejandro Rodríguez Rodríguez 19014181

30/04/2024

Índice

Introducción y propósito del proyecto.

Este documento tiene el propósito de describir el proceso de creación por el que pasó el proyecto de alineamiento de múltiples secuencias, explicar su funcionamiento y ofrecer una conclusión sobre lo aprendido a lo largo de este proyecto.

El objetivo de este proyecto es la creación de un programa que sea capaz de tomar una cantidad *n* de secuencias genéticas, de *m* longitud y alinearlas mediante la introducción aleatoria de guiones (gaps), lo cual permitirá alinear las secuencias mencionadas, facilitando la visualización y el análisis de similitudes entre ellas. Aunque la premisa del proyecto puede parecer sencilla, en realidad existen más factores que influyen en el proceso de alineamiento, los cuales se explorarán en los siguientes apartados.

A lo largo del documento, abordaremos la investigación y los antecedentes recopilados durante los meses de trabajo en el proyecto. También se incluirá pseudocódigo y los enfoques utilizados para resolver el problema del alineamiento. Una vez entendido el proceso de investigación y planificación para crear el programa, se mostrarán fragmentos del código con sus respectivas explicaciones, para finalmente presentar los resultados obtenidos y las conclusiones alcanzadas con este proyecto.

Antecedentes del código e investigación.

Para iniciar el desarrollo de este proyecto primero se llevó a cabo una investigación respecto a las diferentes estrategias que existen para el alineamiento de múltiples secuencias. Esta investigación esta totalmente detallada dentro del documento "*PseudoCodigo_Objetivo1"* pero aquí veremos de manera resumida todos los detalles de esa investigación.

Dentro de las diferentes estrategias existentes para la resolución de las MSA pudimos indagar en las siguientes estrategias:

- MSA y DCA
- Algoritmo de construcción progresiva.
- Aproximaciones iterativas (Método SAGA)

MSA y DCA

La estrategia MSA y DCA esta basado en la programación dinámica. El algoritmo de programación dinámica utiliza una matriz N-dimensional y suponiendo que una secuencia tiene 300 caracteres eso significa que dentro de esta matriz existen 300^N operaciones. Debido a esto, necesita una gran cantidad de recursos computacionales y mucho tiempo. En la práctica apenas se utiliza, solo si $N = 3$ o si las secuencias son cortas, $6 < N < 8$.

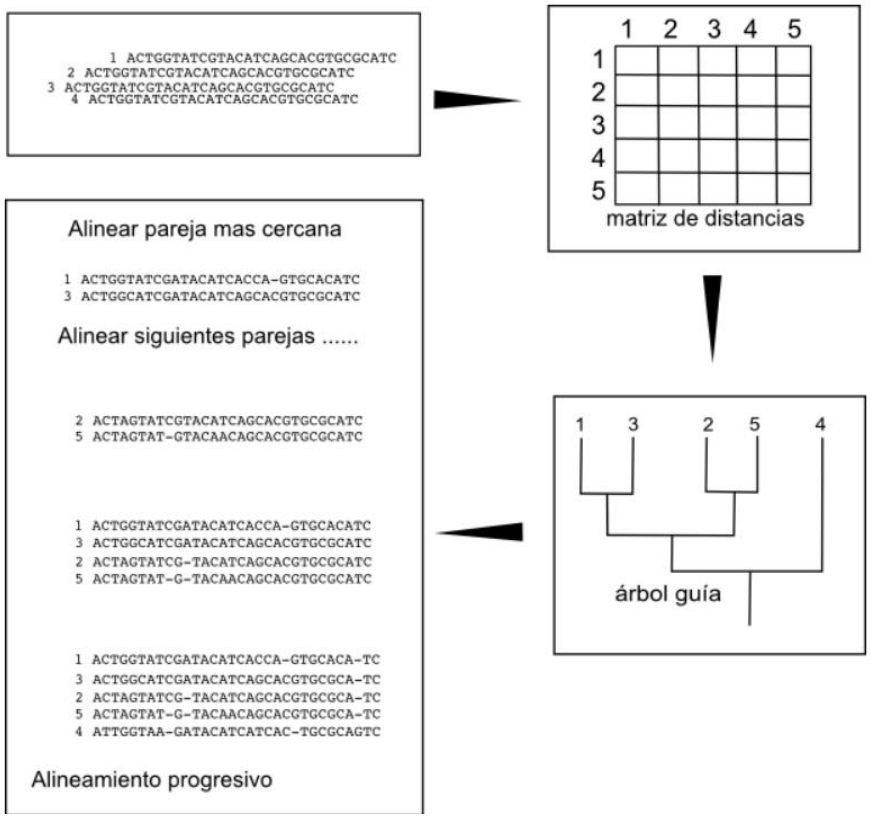
Por su parte, la estrategia DCA (Divide y vencerás) consiste en dividir cada secuencia en dos subsecuencias en un punto cercano al medio. El proceso se repite hasta que las secuencias son lo suficientemente cortas (según un umbral fijado L). A partir de estas subsecuencias se realizan alineamientos por medio del algoritmo de programación dinámica, y después se concatenan para generar el AMS de las secuencias original

Algoritmo de construcción progresiva

La mayoría de los programas relacionados al tema se basan en esta estrategia; El método consiste en primero realizar alineamientos de dos en dos. A partir de estos alineamientos se construye una matriz de distancias entre las secuencias y un árbol guía basado en estas distancias. Mediante este árbol podemos encontrar las parejas de secuencias más similares.

Al alineamiento del par de secuencias más similar se va añadiendo el resto de las secuencias o alineamientos por el orden determinado por el árbol guía.

Nota importante: No nos garantiza que el alineamiento sea el mejor posible, pero es capaz de encontrar una solución óptima eficaz.



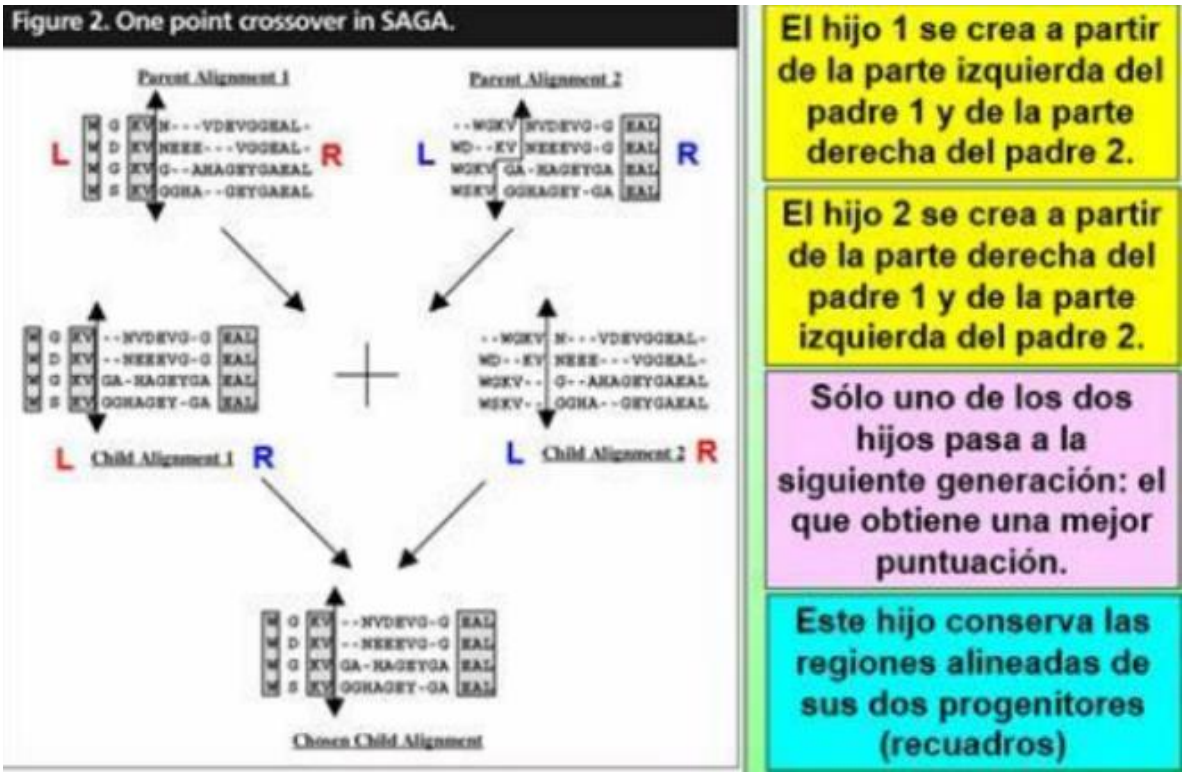
Aproximaciones iterativas, método SAGA

Esta estrategia calcula una solución subóptima mediante un alineamiento progresivo, luego de esto modifican el alineamiento mediante programación dinámica hasta que la solución converge. En otros alineamientos de progresión normal al cometer un error ya no se puede corregir, pero este método soluciona esto. Este consiste en 4 etapas que tratan de emular la selección natural.

1. *Inicialización*: Se escriben las secuencias (una en cada fila), y cada una se desplaza hacia la derecha un número aleatorio de posiciones. Se incluyen huecos en los extremos para que al final todas tengan la misma longitud.
2. *Evaluación mediante una función objetiva*: Se valora cada secuencia siguiendo un criterio. Cuanto mejor es el alineamiento, mayor es la puntuación. A partir de los alineamientos iniciales se obtiene la "descendencia" esperada. De las secuencias iniciales, se mantiene el 50% en la siguiente generación G1, la otra mitad se sustituye por la descendencia de los alineamientos más aptos (según el criterio seleccionado).
3. *Reproducción (mutaciones y recombinaciones):* Para generar la descendencia, se modifican los alineamientos parentales con mutaciones o recombinaciones, siguiendo la función objetiva, y el 50% con mayor puntuación es el que pasa a la siguiente generación. Las mutaciones (introducción de un hueco, no cambio de aminoácidos) no cambian la secuencia porque se arruinaría el alineamiento. Para introducir los huecos se generan primero dos subgrupos y en cada grupo se añaden huecos en dos posiciones aleatorias. A estos huecos se les pueden realizar más modificaciones, como desplazar los dos, o dividirlos en vertical/horizontal y mover sólo uno de ellos. Cuando hay recombinación se necesitan dos progenitores, a partir de los cuales se genera

un primer hijo con la parte izquierda del parental 1 y la parte derecha del parental 2 y el segundo hijo con la parte derecha del parental 1 y la parte izquierda del parental 2.

4. *Finalización:* La señal de parada del algoritmo depende del criterio fijado, que puede ser una puntuación o un número determinado de generaciones (normalmente 100).



Planteamiento del programa y seudocódigo.

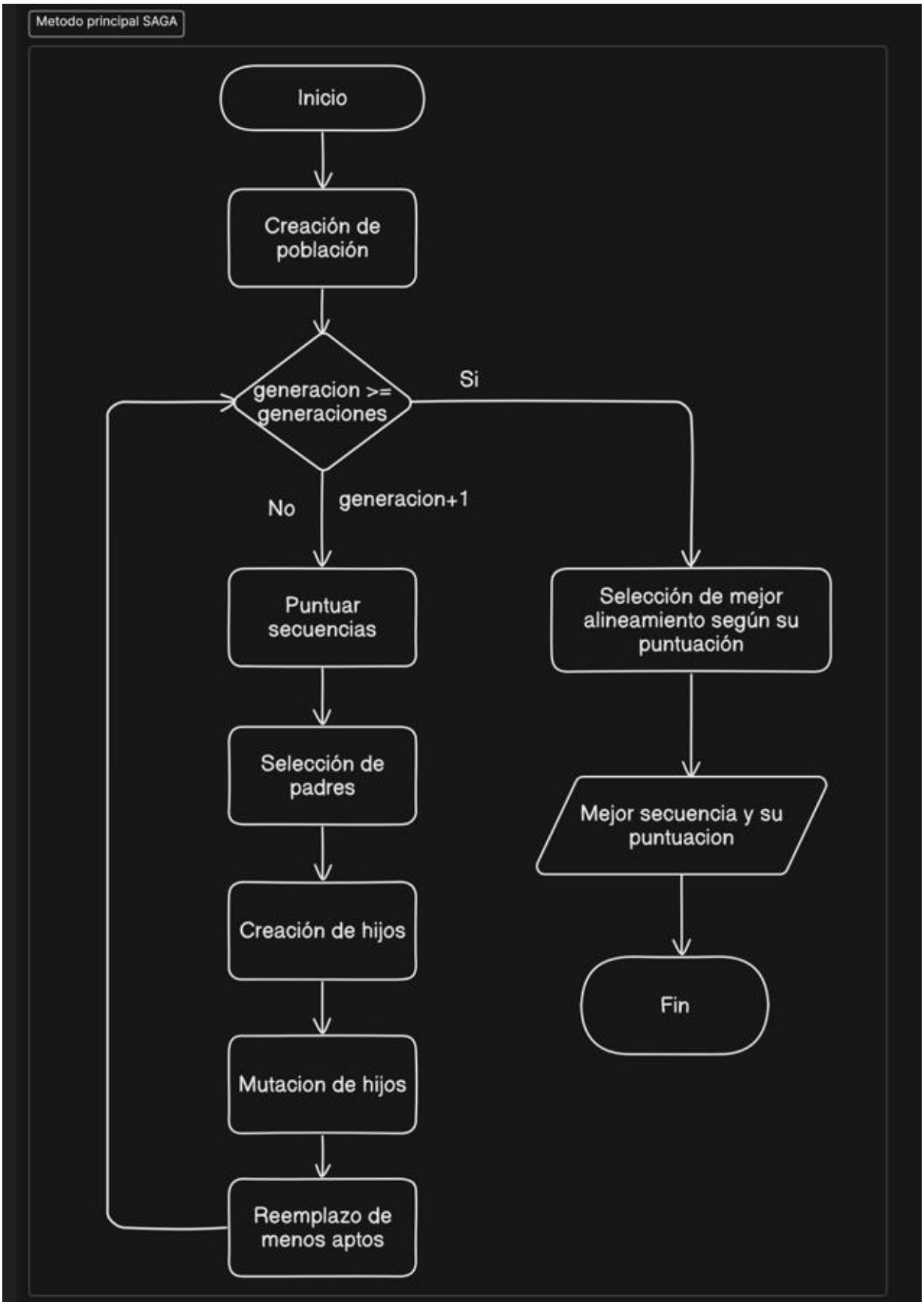
Luego de la investigación realizada llegue a la conclusión de que el método de MSA no es conveniente para el proyecto debido a la limitación que presenta, por otra parte, el DCA parecía prometedor, pero no se asemejaba al ejemplo visto durante el planteamiento del proyecto. El algoritmo de construcción progresiva también se veía prometedor pero debido a lo complejo que suena programarlo no se usara. Por último, el seleccionado fue el método iterativo SAGA, esto debido a su fácil comprensión y a su gran similitud al planteamiento visto durante la exposición del proyecto.

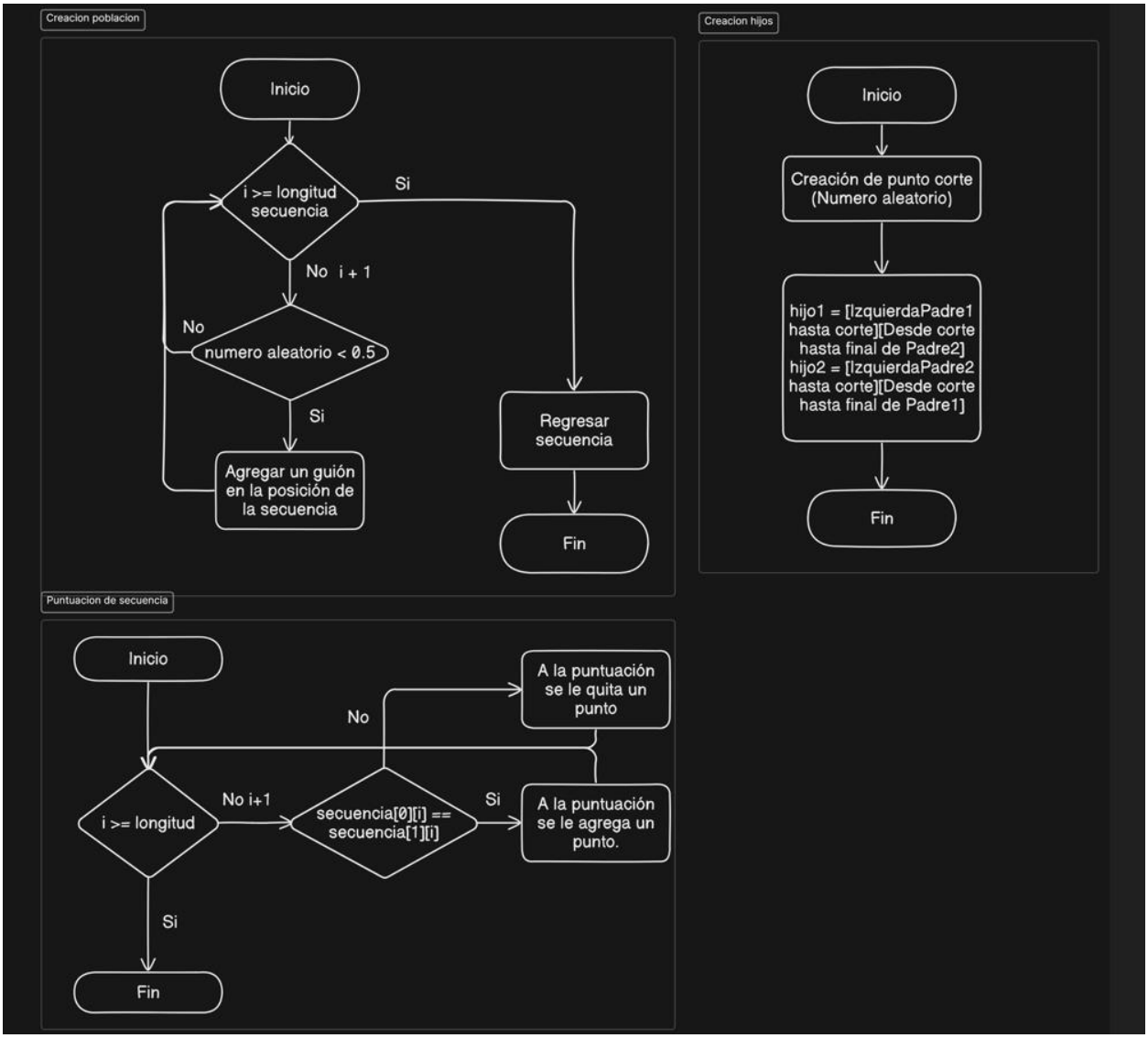
Una vez decidida la estrategia con la que se apoyó este proyecto se empezó la creación de un seudocódigo el cual tuvo muchas limitantes practicas además de no implementar funciones que se verán a continuación. Aquí un fragmento simplificado del seudocódigo.

Fragmento del seudocódigo y diagramas

Proceso AlineamientoSAGA #Metodo principal

- Inicialización de secuencias, tamaño de población, rango de mutación y generaciones.
- La población se genera en base a un método el cuál se repite en un ciclo hasta alcanzar el tamaño de esta.
- Se inicia el ciclo principal de SAGA que se repetirá según el número de generaciones especificadas. (Para generación hasta generaciones hacer)
    - Se consiguen las puntuaciones de la población.
    - Se seleccionan los padres en base a las puntuaciones más altas.
    - Se crean dos hijos usando el método cruza usando a los padres seleccionados.
    - Cada hijo muta.
    - Se reemplazan los menos aptos de la población usando a los hijos.
- Una vez termina el ciclo principal se toma al mejor alineamiento en base a su puntuación para al final mostrarlo junto su puntuación.


Metodo principal SAGA
Inicio
Creación de población
generacion >= generaciones
Si
No
generacion+1
Puntuar secuencias
Selección de padres
Creación de hijos
Mutacion de hijos
Reemplazo de menos aptos
Selección de mejor alineamiento según su puntuación
Mejor secuencia y su puntuacion
Fin



Problemas observados y corregidos

Esa es la idea inicial que se tenía para el código, pero mientras este se desarrollaba y probaba salieron a relucir diferentes complicaciones que tuvieron que ser resueltas directamente en el código. Estas son algunas de las complicaciones que se resolvieron posterior al seudocódigo.

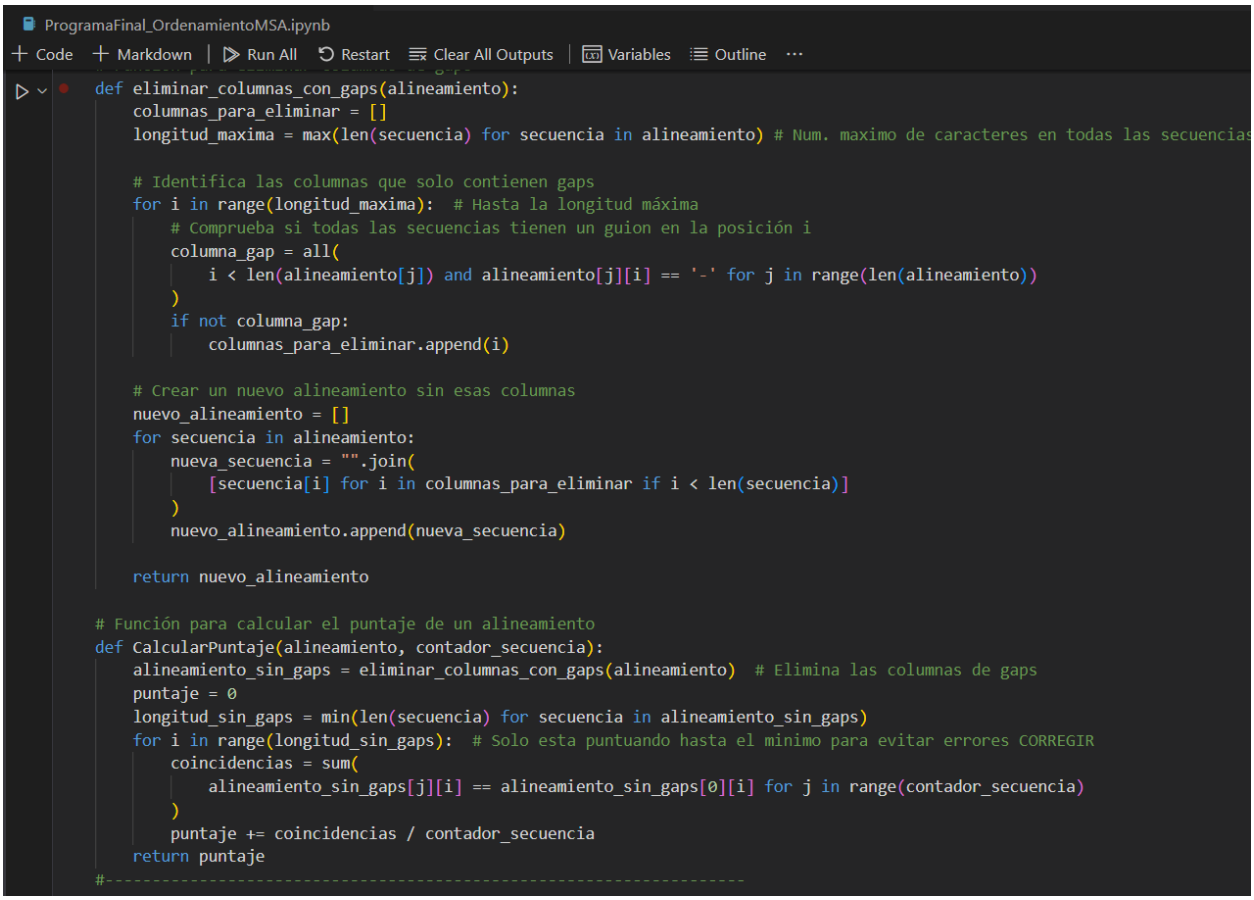
1. *Eliminación de columnas de gap's:* Cuando se planteo la idea del proyecto se había mencionado que el algoritmo debería ser capaz de identificar si existían columnas de gap's y eliminarlas de las secuencias, esto se corrigió mas adelante y se agrego la función en el método de puntuación.
2. *Limitaciones de 2 secuencias:* Al realizar el código se pudo observar que este solo podía alinear dos secuencias por lo que se adapto permitiendo introducir *n* cantidad de secuencias.
3. *Zigzagueo de gap's al crear hijos:* Una observación hecha por el profesor era que los cortes realizados no distinguían si el ultimo carácter era una letra o un gap, por lo que se adaptó el código para que reconociera si el ultimo carácter del corte era un gap para saltar hasta la siguiente letra.

4. *Partición de secuencias en 3:* Como se puede observar originalmente los 2 hijos se creaban en base a la creación de solo un punto de corte que dividiera las secuencias, esto se corrigió para que se crearan 2 puntos de corte para que así, por ejemplo, el hijo 1 tuviera la izquierda del padre 1, el centro del padre 2, y el lado derecho del padre 1.

Código y funcionamiento.

En este apartado se mostrará fragmentos del código final, se explicará su funcionamiento y algunos datos surgidos durante la elaboración de este.

Puntuación de secuencias.

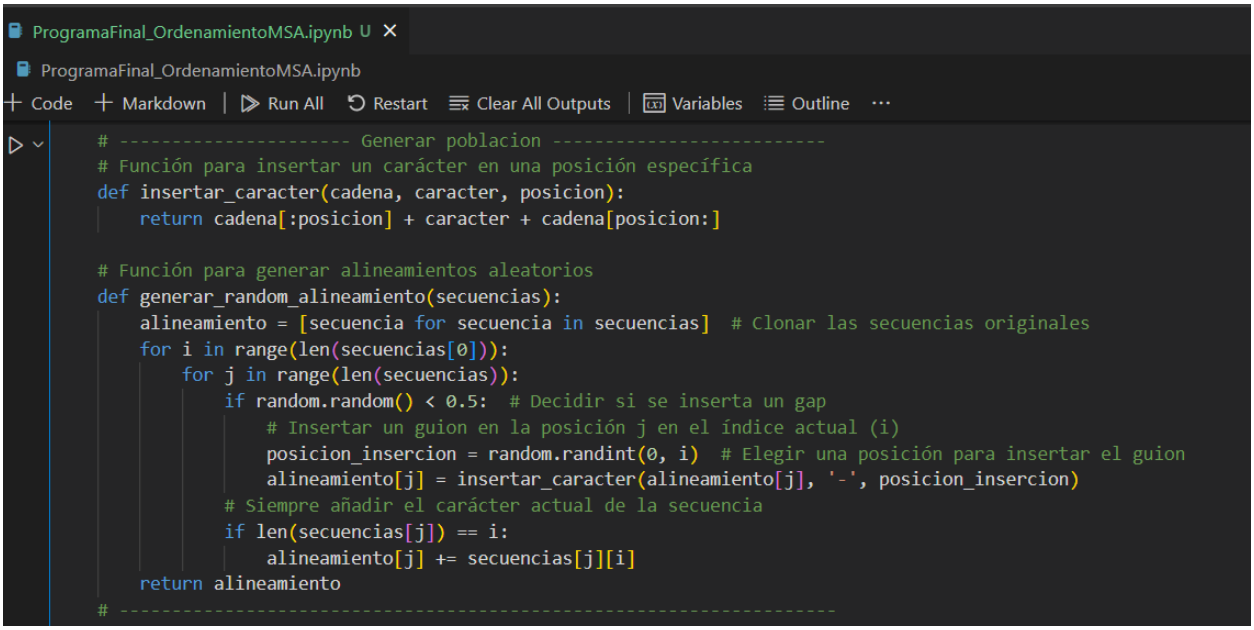


En esta sección existen 2 métodos, uno encargado para eliminar las columnas de gap's y el otro para calcular el puntaje.

El método de eliminación de columnas de gap's lo que realiza es un nuevo arreglo de secuencias donde se hayan eliminado estas columnas de gap's, para ello consigue la longitud máxima de las secuencias y va recorriendo a través de i cada columna de las secuencias. Dentro de este ciclo comprueba si todas las secuencias tienen un gap en la posición i y si es así se agregan en las columnas a eliminar. Posterior a este ciclo viene otro en donde se crea una nueva secuencia sin estas columnas y se regresa el nuevo alineamiento.

El método de calcular puntaje es mas sencillo, en este primero se adapta las secuencias a utilizar con el método anterior, posterior a eso se toma la longitud más pequeña para así recorrer un ciclo e ir columna por columna calculando las coincidencias que existen en las filas, para al final calcular el puntaje sumándole las coincidencias divididas entre el contador de secuencias.

Generar población.



En este apartado existen dos métodos, el primero se encarga de la inserción del carácter mientras que el segundo es el encargado de generar estos alineamientos aleatorios.

La función del primer método es la de tomar la cadena, el carácter a insertar y la posición donde se va a insertar, regresando así la nueva cadena con el carácter (En este caso el gap) insertado recorriendo la secuencia y sin reemplazar ninguno de los caracteres.

El segundo método toma el arreglo de secuencias y lo clona en un nuevo arreglo, posterior a eso usa un ciclo para recorrer cada columna hasta alcanzar la longitud de la primera secuencia, luego de eso se inicializa otro ciclo que recorre cada fila (secuencia), dentro de este ultimo ciclo se genera un numero aleatorio y si es menor a 0.5 entonces se inserta un gap en la posición y alineamiento generado por otro numero aleatorio, este gap se agrega al clon de la secuencia original. En caso de que no se inserte gap solamente se copia el carácter de la secuencia original.

Cruza y mutación.

```
# ---------- Cruza y mutacion de hijos ----------
# Función para ajustar el punto de corte (Para zigzaguear el guion)
def ajustar_punto_corte(secuencia, punto_corte):
    while punto_corte > 0 and secuencia[punto_corte - 1] == '-':
        punto_corte += 1  # Mover el punto de corte a la derecha
        if punto_corte >= len(secuencia):
            break
    return punto_corte
```

```python
# Función para cruzar los alineamientos
def Cruza(padre1, padre2):
    longitud = len(padre1[0])

    # Elegir dos puntos de corte
    punto_corte1 = random.randint(0, longitud - 1)  # Primer punto de corte
    punto_corte2 = random.randint(punto_corte1 + 1, longitud)  # Segundo punto de corte

    # Ajustar los puntos de corte
    punto_corte1 = ajustar_punto_corte(padre1[0], punto_corte1)
    punto_corte2 = ajustar_punto_corte(padre1[0], punto_corte2)
    hijo1 = []
    hijo2 = []

    for j in range(len(padre1)):
        # Crear partes para el hijo 1
        izquierda_hijo1 = padre1[j][:punto_corte1]  # Parte izquierda de padre1
        medio_hijo1 = padre2[j][punto_corte1:punto_corte2]  # Parte media de padre2
        final_hijo1 = padre1[j][punto_corte2:]  # Parte final de padre1
        hijo1.append(izquierda_hijo1 + medio_hijo1 + final_hijo1)

        # Crear partes para el hijo 2
        izquierda_hijo2 = padre2[j][:punto_corte1]  # Parte izquierda de padre2
        medio_hijo2 = padre1[j][punto_corte1:punto_corte2]  # Parte media de padre1
        final_hijo2 = padre2[j][punto_corte2:]  # Parte final de padre2
        hijo2.append(izquierda_hijo2 + medio_hijo2 + final_hijo2)
    # print('Parte izquierda hijo1: ',izquierda_hijo1,
    #           "\nParte media hijo1:",medio_hijo1,
    #           "\nParte final hijo1:",final_hijo1)
    return hijo1, hijo2
```

```python
# Función para aplicar mutación a un alineamiento
# Es literalmente la misma función que la de generar el alineamiento random, con la diferencia de que recorre la
# longitud maxima y recibe un valor de mutacion lo que se puede modificar desde fuera.
def mutacion(alineamiento, valor_mutacion):
    mutacion_alineamiento = ['' for _ in range(len(alineamiento))]  # Alineamiento vacío
    longitud_maxima = max(len(secuencia) for secuencia in alineamiento) # Longitud maxima
    for i in range(longitud_maxima):
        for j in range(len(alineamiento)):
            if i < len(alineamiento[j]) and random.random() < valor_mutacion:
                posicion_insercion = random.randint(0, i)  # Posición aleatoria
                mutacion_alineamiento[j] = insertar_caracter(mutacion_alineamiento[j], '-', posicion_insercion)
                mutacion_alineamiento[j] += alineamiento[j][i]
    return mutacion_alineamiento
# ----------------------------------------------------------------------------------------
```

Este es el apartado más largo, pero a la vez menos complicado de explicar. Este consiste en 3 métodos que son los siguientes…

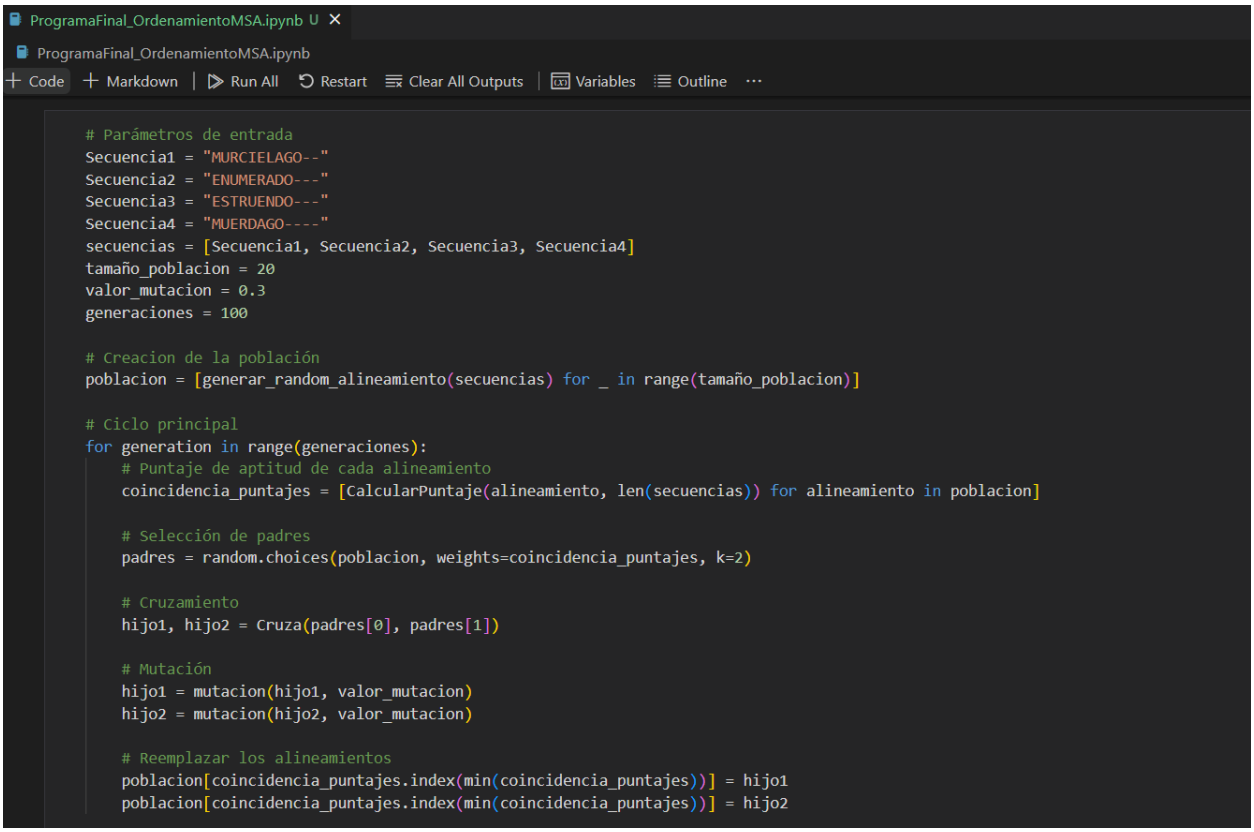
Ajustar el punto de corte se encarga detectar si el ultimo carácter de un punto de corte es un gap o una letra, en caso de ser un gap recorre el punto de corte un espacio a la derecha.

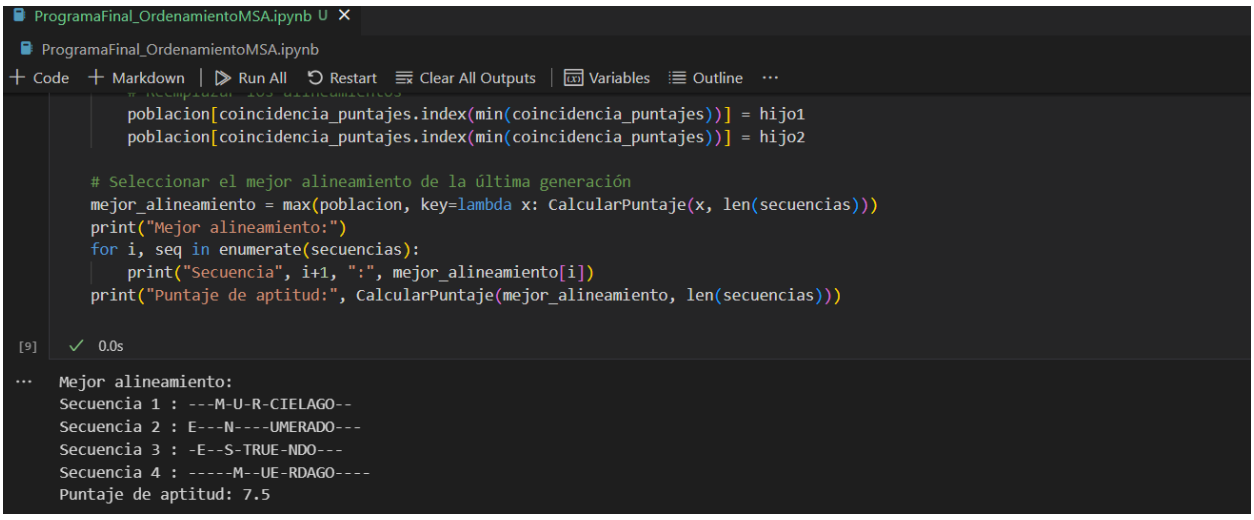
Cruza se encarga de generar las cruzas entre los dos padres y crear los dos hijos, en este se toma la longitud de la secuencia 1 para generar los dos puntos de corte a partir de un generador de números aleatorios, luego estos números son evaluados con la función anterior y adaptados en caso de que su ultimo carácter si sea un gap. Después de esto se inicia un ciclo que recorre cada una de las filas y toma

como ya se había comentado la parte izquierda, central y derecha para cada uno de los hijos, para al final juntarlas en una misma secuencia, crear a los hijos y regresarlos.

Mutación hace prácticamente lo mismo que el método de generación de población con la única diferencia de que se proporciona un valor de mutación para saber si se inserta o no se inserta gap.

Ciclo principal.





En este únicamente se dan los parámetros de entrada tales como las secuencias, el tamaño de la población, valor de mutación y generaciones. Posterior a eso se llama el método de generar población el cuál se mete en un ciclo según el tamaño de esta.

Luego de esto se inicia el ciclo principal que consta en sacar las puntuaciones de toda la población (también se usa un ciclo), luego se seleccionan los padres con el método de random.choices que toma a los que mayor puntuación tienen, se hacen los hijos llamando al método de cruza, estos mutan y por último se reemplazan los dos de puntaje mas bajo con los dos hijos generados, y este proceso se repite *n* numero de generaciones.

Por último, se selecciona el mejor alineamiento basados en su puntuación y se imprime junto a su puntuación.

Nota: No se ve en estas capturas, pero se agregó unos prints para imprimir las secuencias originales y los hijos de cada generación.

Resultados del código.

A continuación, se mostrarán diferentes resultados que nos arrojó el código y se comentara un poco de ellos.

Palabras de prueba.

```
Mejor alineamiento:
Secuencia 1 : ---M-U-R-CIELAGO--
Secuencia 2 : E---N----UMERADO---
Secuencia 3 : -E--S-TRUE-NDO---
Secuencia 4 : -----M--UE-RDAGO----
Puntaje de aptitud: 7.5
```

Para esta prueba se usaron las palabras “Murciélago, Enumerado, Estruendo y Muérdago” como secuencias, podemos observar que se consiguió un puntaje del 7.5 y podemos observar el alineamiento de varios caracteres. Vemos el alineamiento de caracteres como las O del final, las G anteriores, las A, las D, las dos E y por último la U. Aunque es muy probable que este alineamiento se pueda mejorar incluso más hubo un considerablemente buen alineamiento.

Secuencias de prueba.

```
Mejor alineamiento:
Secuencia 1 : G-----C---------T---------AG-C--TAGC-TA-GC-T-AGC-TAGCTGATCGTAGACTGCTAGCTAGCTAGCTAGCTGATCG
Secuencia 2 : ---------G--C--------TG--ATC--G-T--AGACT-G-CTA-GCTAGCTAGCTAGCTAGCTAGCTAGTCGATCGTAGACTGC
Secuencia 3 : --G-C----T-A-----GCT--GA-CTA--GCT-A-GC-TAGCTAGCTAGTCG-ATCGTAGACTAGCTAGCTAGCTAGCT
Secuencia 4 : ---G-CT-A-----G---C-TAGCTA-G-CT-A--GCT-A--GCTAGC-TAG-C-TAGCTAGCTAGCTAGTCGATCGTAGCTG
Puntaje de aptitud: 37.25
```

+ Code + Markdown

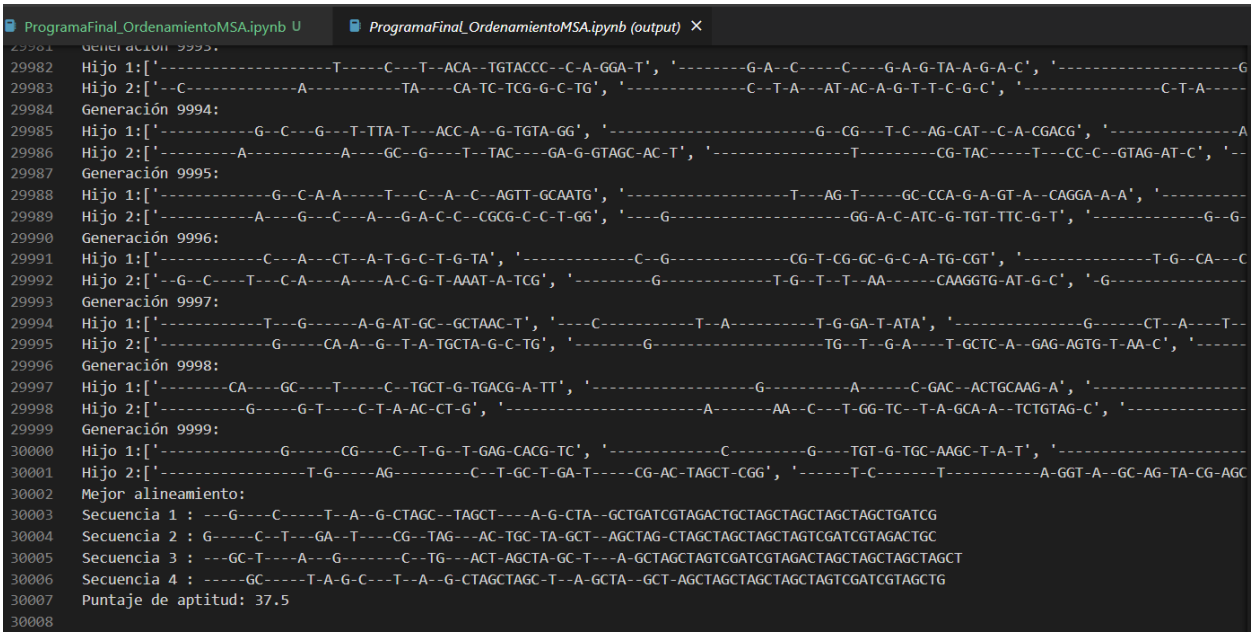
Las secuencias usadas para esta prueba fueron las siguientes…

```
Secuencia 1: GCTAGCTAGCTAGCTAGCTAGCTGATCGTAGACTGCTAGCTAGCTAGCTAGCTGATCG
Secuencia 2: GCTGATCGTAGACTGCTAGCTAGCTAGCTAGCTAGCTAGCTAGTCGATCGTAGACTGC
Secuencia 3: GCTAGCTGACTAGCTAGCTAGCTAGCTAGTCGATCGTAGACTAGCTAGCTAGCTAGCT
Secuencia 4: GCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGCTAGTCGATCGTAGCTG
```

En esta prueba ya se complica más ver las coincidencias, aunque si es posible distinguir algunas, pero se hace evidente algunos problemas tales como que las columnas de gaps no se están eliminando en su

totalidad y también que no se esta dividiendo correctamente al momento de hacer la cruza, esto se abordara a detalle en las conclusiones.

Prueba de las 10,000 generaciones.



En esta prueba podemos observar los resultados del programa luego de 10,000 generaciones con una población de 50. Estos comparten cierta similitud con los presentados anteriormente.

Conclusiones y comentarios finales.

Para finalizar este reporte podemos destacar como el programa logro concretar casi completamente el propósito de este ya que es capaz de alinear secuencias de *m x n* tamaños, aunque los resultados son medianamente correctos me parece importante señalar algunos detalles que se hicieron evidentes al concretar este, con esto me refiero a algunos puntos que se pueden mejorar.

- Aunque la secuencia alineada al final no presenta tantas columnas de gaps gracias al método implementado en la parte de puntuación es cierto que se podría implementar nuevamente el método de eliminación de columnas de gaps al finalizar una generación para así reducir más estas columnas.
- La puntuación de los alineamientos, aunque no está equivocado aun así se podría mejorar mas si se agregan mas condiciones tales como que se reduzcan medios puntos por discordancias en el alineamiento
- La creación de hijos y cruza de los padres al final, aunque también funciona no está generando puntos de cortes tan convenientes para la mutación de estos ya que siento que al ser un punto aleatorio el primer corte puede abarcar mucha longitud de la secuencia o muy poca.
- En secuencias largas por alguna razón la mayoría de los gaps se están concentrando en el inicio de las secuencias, cosa que se podría corregir revisando y mejorando la mutación y la generación aleatoria de secuencias.

Aun con estos detalles el programa logra cumplir de manera satisfactoria el alineamiento además de poder continuar durante un gran numero de generaciones con una población también grande.