

MÉTODOS PARA EL ALINEAMIENTO MÚLTIPLE DE SECUENCIAS BASADOS EN APRENDIZAJE REFORZADO.

T E S I S

Que para obtener el grado de
Maestro en Cómputo Estadístico

Presenta

Ángel Adrián Domínguez Lozano

Director de Tesis:

Dr. José Ulises Márquez Urbina

Autorización de la versión final

Dedico esta tesis a mi madre por siempre guiarme y quererme, a mi padre que me enseño el valor del trabajo y a mi abuelito por siempre haber creido en mí.

Resumen

El problema del **alineamiento múltiple de secuencias** (MSA, por sus siglas en inglés) consiste en acomodar sucesiones de ADN, ARN o aminoácidos en un arreglo que permita identificar regiones de similitud para hacer inferencias evolutivas o funcionales entre ellas.

En este trabajo se exploran tres enfoques estado-acción-recompensa para la creación de alineamientos utilizando técnicas de **aprendizaje reforzado** (RL, por sus siglas en inglés). Se realizan comparaciones con dos de los métodos más usados actualmente, CLUSTALW y MAFFT. Los resultados indican que los métodos basados en RL considerados son competitivos en cuanto a calidad del alineamiento y al coste computacional.

Palabras clave: alineamiento múltiple de secuencias, aprendizaje reforzado, alineamiento progresivo, redes neuronales, bioinformática

Agradecimientos

A mi madre y a mi hermano por siempre estar conmigo.

A mi asesor el Dr. Ulises por aconsejarme y guiarme en la realización de esta tesis.

A toda la comunidad del CIMAT que me permitió desarrollar mis estudios en las mejores condiciones.

Al CIMAT por proveerme de nuevos retos y conocimientos.

Al CONACYT por otorgarme apoyo económico durante mis estudios de maestría (CVU 927734).

Índice general

Resumen	III
Agradecimientos	V
1. Introducción	1
2. Preliminares	7
2.1. Alineamiento de dos secuencias	8
2.1.1. Cantidad de alineamientos	13
2.1.2. Algoritmos para alinear dos secuencias	15
2.2. Alineamiento múltiple de secuencias	19
2.2.1. CLUSTALW	23
2.2.2. MAFFT	23
2.2.3. MUSCLE	25
2.2.4. T-Coffee	27
2.3. Aprendizaje reforzado	29
2.3.1. Deep Q-Learning	34
2.3.2. Neural Fitted Q Iteration	34
2.4. Enfoques existentes de RL para MSA	37
3. Enfoques de RL propuestos	41
3.1. Alineamiento Progresivo sobre el espacio de permutaciones de secuencias	42
3.2. Alineamiento Progresivo sobre el espacio de árboles filogenéticos	43
3.3. Extensión de NW sobre el espacio de sufijos de secuencias	51
3.4. Comparaciones entre los enfoques	52
4. Análisis del rendimiento de los enfoques propuestos	57
4.1. Datos	58
4.2. Deep Q-Learning con enfoques basados en alineamiento progresivo	60
4.2.1. Comparación con MAFFT y CLUSTALW	63
4.2.2. Comparación contra enfoques de Mircea y Jafari	63
4.2.3. Comparación OXBENCH	74
4.3. Métrica GUIDANCE	78
4.4. NFQ	84
5. Conclusiones y trabajo futuro	89

Referencias	91
A. Resultados OXBENCH	95
B. Detalles de la implementación	121
C. Código	123

Capítulo 1

Introducción

Durante la segunda mitad del siglo pasado el poder de cómputo creció en gran medida. Muchos campos del conocimiento fueron revolucionados debido a este crecimiento, y la biología no fue la excepción. De esta forma, aunado con el descubrimiento de la estructura del ADN, durante el siglo XX se creó la bioinformática. Pevsner (2015) define la bioinformática como “El uso de bases de datos informáticos y algoritmos computacionales para analizar proteínas, genes y la colección completa del ADN que comprende un organismo (el genoma).” En las últimas décadas se han desarrollado sistemas y algoritmos que utilizan el creciente poder computacional para expandir las fronteras del conocimiento biológico, particularmente el genómico.

Un área de estudio de gran importancia en la bioinformática es el análisis de secuencias de ADN, RNA o proteínas que se realiza para determinar diversos aspectos de lo que codifican; por ejemplo, identificar regiones de similitud entre secuencias ayuda a realizar inferencias sobre la funcionalidad de los genes que codifican (Pevsner, 2015). Este análisis tiene aplicaciones muy importantes en la investigación biológica y en el área médica.

A finales del 2019 (WHO, 2020) surgió la pandemia del COVID-19, enfermedad causada por el virus SARS-CoV-2 , esto ocasionó afectaciones al sistema de salud y a la economía en prácticamente todos los países del mundo, por lo que su estudio es de

vital importancia. Se han llevado a cabo recientemente trabajos para determinar el origen del virus SARS-CoV-2 ([Zhang, Wu, y Zhang, 2020](#)) al encontrar su lugar en el árbol genealógico de otros virus, determinando su similaridad contra ellos; y para genotipar el virus observando en distintas muestras las mutaciones más predominantes durante la pandemia con el objetivo de desarrollar una vacuna ([Yin, 2020](#)). Un primer paso para realizar los estudios antes mencionados, es llevar a cabo un *alineamiento múltiple de secuencias* (MSA, por sus siglas en inglés).

Consideremos primero que se quiere alinear dos secuencias, este problema consiste en agregar algunos caracteres denominados gaps a las secuencias de tal forma que los gaps introducidos representen mutaciones (eliminaciones o inserciones) que sucedieron desde una secuencia ancestro común de las dos secuencias en cuestión. Para alinear más de dos secuencias, se consideran las mismas condiciones, es decir de agregan gaps a todas las secuencias de tal forma que al final todas las secuencias tengan la misma longitud y, en conjunto, el alineamiento represente el historial de mutaciones que dio origen a las secuencias. Los alineamientos generados sirven como insumo para alguna aplicación que así lo requiera.

De acuerdo a [Ewens y Grant \(2001\)](#), existen tres características binarias que distinguen a los distintos tipos de alineamiento :

- Con respecto a la longitud de las secuencias que se requiere sean alineadas, se considera alineamiento *global* si se debe considerar la longitud total de la secuencia. Se considera alineamiento *local* si es de interés solo alinear subsecuencias de las secuencias de entrada.
- Si el alineamiento permite la representación de eliminaciones e inserciones al colocar gaps entre los elementos de las secuencias, el alineamiento es *gapped*. En caso contrario se tiene un alineamiento *ungapped*.
- Si se alinean dos secuencias, se tiene un alineamiento *pairwise*. Cuando se alinean

más de dos secuencias se tiene un alineamiento *múltiple*.

En este trabajo, se abordará el problema de encontrar *alineamientos gapped, múltiples y globales* de secuencias de ADN que maximicen un SP-score (ver Capítulo 2). Este problema es especialmente complejo (Pevsner, 2015) debido a que la cantidad de posibles alineamientos crece combinatoricamente con respecto a la cantidad de secuencias y a sus longitudes. Más aún, la complejidad del algoritmo que obtiene la solución óptima usando programación dinámica es $O(2^N L^N)$, con N la cantidad de secuencias a alinear y L una cota superior de las longitudes de las secuencias. En consecuencia, diversos métodos heurísticos se han desarrollado, como CLUSTALW, MAFFT, MUSCLE, T-COFFEE. Todos estos métodos, por su naturaleza no exacta, mantienen un compromiso entre velocidad y calidad del alineamiento. Es por esto que en la actualidad se siguen proponiendo métodos que conlleven a un mejor equilibrio entre dichos aspectos.

Es de particular interés el alineamiento progresivo, pues este es la base de algunos de los métodos más utilizados. En el alineamiento progresivo, se realizan secuencialmente alineamientos entre alineamientos previamente generados, de tal forma que inicialmente se tienen N secuencias disjuntas y al final se tiene un solo alineamiento con las N secuencias integradas en un alineamiento. El proceso por el cual se van integrando las partes disjuntas suele representarse mediante un árbol. En este trabajo de tesis se proponen y se exploran métodos heurísticos basados en aprendizaje reforzado (RL, por sus siglas en inglés) para la obtención de alineamientos con un SP-score grande y sin utilizar poder de cómputo excesivo o inviable.

En palabras de Knuth (1997) “Las computadoras han sido tradicionalmente asociadas a la solución de problemas numéricos, ... Desde los 60’s, sin embargo, las computadoras se han utilizado con mas frecuencia para problemas donde los números están presentes solo por coincidencia; Se están utilizando las capacidades de toma de decisiones de la computadora, en lugar de su capacidad para hacer operaciones aritméticas.” Esta capacidad de toma de decisiones, se ha utilizado para consolidar

al aprendizaje reforzado. Según Sutton (1998) “El aprendizaje reforzado se refiere al problema de un agente de aprendizaje que interactúa con su entorno para lograr un objetivo. En lugar de recibir ejemplos del comportamiento deseado, el agente debe descubrir por ensayo y error cómo comportarse para obtener la mayor recompensa.” Recientemente, esta estrategia se ha utilizado para generar sistemas capaces de replicar algunas tareas con desempeño mejor que el humano. Por ejemplo, en cuestión de juegos (Silver y cols., 2018), se propone un algoritmo capaz de aprender a jugar ajedrez, shogi y Go mediante episodios de auto exploración, es decir que el algoritmo aprende jugando contra el mismo. Otra de las aplicaciones de vanguardia (Kiran y cols., 2020) es la relacionada con la automatización de algunas labores, como el desarrollo de vehículos autónomos.

Sutton (1998) menciona que se pueden identificar tres etapas en el desarrollo histórico del aprendizaje reforzado. En la primera etapa, que comprende el período anterior a 1985, se desarrolló la etapa de ensayo y error. La segunda etapa, que va de 1985 a principios de este siglo, se caracterizó por enfocarse en el uso de funciones de valor. Finalmente, Sutton predijo que el futuro del RL se enfocaría a la investigación de estructuras que permitirán una correcta estimación de las funciones de valor. Esto actualmente se está cumpliendo, ver por ejemplo (Mnih y cols., 2013), pues recientemente se han utilizado estructuras basadas en redes neuronales para estimar funciones de valor.

Por tanto, en la actualidad el aprendizaje reforzado se consolida como una alternativa para problemas complejos que se puedan modelar como un proceso de toma de decisiones. Esta tesis tiene como objetivo explorar métodos basados en aprendizaje reforzado para solucionar el problema del alineamiento múltiple de secuencias y de esta forma contribuir con los esfuerzos que diversos autores han realizado, analizando cómo la estrategia del aprendizaje reforzado se desempeña en este problema complejo de la bioinformática. Ramakrishnan, Singh, y Blanchette (2018), Ramakrishnan (2018) proponen un enfoque que requiere un intenso poder computacional, pues des-

de una propuesta de alineamiento se exploran movimientos entre gaps y elementos de las secuencias hasta que se cumpla un criterio de paro. [Mircea, Bocicor, y Dîncu \(2014\)](#), [Mircea, Bocicor, y Czibula \(2016\)](#) y [Jafari, Javidi, y Rafsanjani \(2019\)](#) proponen un enfoque basado en alineamiento progresivo restringido a aquellos procesos que se pueden representar mediante una permutación de las secuencias a utilizar.

En este trabajo se proponen 3 métodos y se exploran sus resultados. El primero de ellos basado en una reducción del espacio de búsqueda de las propuestas de [Mircea y cols. \(2014\)](#), [Mircea y cols. \(2016\)](#), [Jafari y cols. \(2019\)](#). El segundo está también basado en alineamiento progresivo, pero explorando sobre todos los posibles procesos sin considerar la restricción de las permutaciones. Finalmente, el tercer método consiste en construir alineamientos columna a columna considerando los sufijos, aún por alinear, de cada secuencia.

El primer capítulo de esta tesis consiste en introducir algunos conceptos importantes para contextualizar el problema y la estrategia a utilizar. En el segundo capítulo se revisa la teoría necesaria para poder llevar a cabo la implementación de los métodos propuestos. El tercer capítulo contiene el análisis del rendimiento de los métodos propuestos mediante comparación con los resultados de [Mircea y cols. \(2014\)](#), [Mircea y cols. \(2016\)](#), [Jafari y cols. \(2019\)](#) y mediante el análisis de algunos alineamientos proveidos en [Carroll y cols. \(2007\)](#). Finalmente en la conclusión se reflexiona sobre los resultados obtenidos y se delinea el posible trabajo a futuro a realizar. Se incluye también en el anexo detalles importantes sobre la implementación realizada así como el código generado.

Capítulo 2

Preliminares

En este capítulo revisaremos teoría básica sobre alineamiento múltiple de secuencias, se revisará la teoría necesaria sobre aprendizaje reforzado que se aplicará en los métodos propuestos y finalmente se analizarán los métodos en la literatura actual sobre métodos basados en RL para resolver MSA. En particular es de interés establecer claramente la definición del problema del MSA, conocer las variantes que tiene para especificar la versión del problema a abordar en este trabajo y también revisar los métodos e ideas que los métodos más utilizados emplean.

En cuanto al aprendizaje reforzado, revisaremos parte de la teoría que respalda algunos algoritmos que utilizaremos en sus casos generales y especificaremos el tipo de ambiente en los que se utilizarán estas técnicas; Esto se hará revisando métodos que en años recientes han utilizado el aprendizaje reforzado en la construcción de políticas que conlleven a la creación de un alineamiento múltiple de secuencias de buena calidad, realizando una búsqueda inteligente en un espacio de búsqueda estado-acción-recompensa determinista pero con crecimiento combinatorio con respecto a su dimensión.

DNA and RNA structure

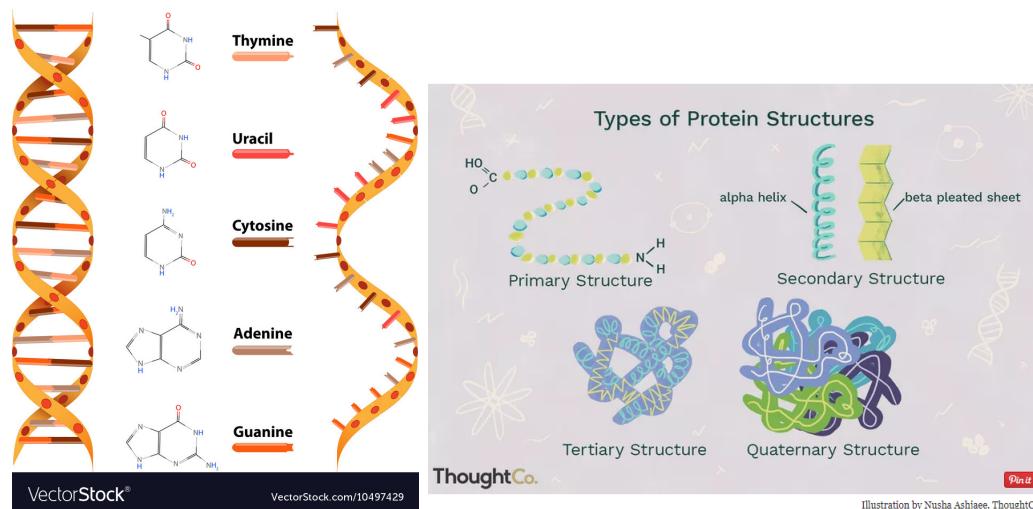


Figura 2.1: Secuencias de ADN, RNA y proteína.

2.1. Alineamiento de dos secuencias

Consideraremos en esta sección secuencias de ADN, ARN y proteínas. Una representación visual de estos objetos se puede apreciar en la Figura 2.1. Cabe destacar que el ADN está conformado por dos hebras, el ARN por solo un hebra y las proteínas tienen una estructura compleja y muy variante, de hecho, en el caso de las proteínas la forma tridimensional que ésta tenga tiene relación con su funcionalidad (Pevsner, 2015).

Las hebras de ADN, ARN y las estructura primaria de las proteínas pueden representarse mediante secuencias de caracteres de un alfabeto \mathbb{Q} , donde cada elemento de \mathbb{Q} representa a un nucleótido, o aminoácido en el caso de las proteínas. Estos son para cada caso:

- ADN: $\mathbb{Q} = \{A, C, G, T\}$, que representan a la adenina, citosina, guanina y timina, respectivamente.
- ARN: $\mathbb{Q} = \{A, C, G, U\}$, que representan a la adenina, citosina, guanina y uracilo, respectivamente.

- Proteínas: $\mathbb{Q} = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$, que representan a la alanina, arginina, asparagina, ácido aspartico, cistina, glutamina, ácido glutámico, glicina, histidina, isoleucina, leucina, lisina, metionina, fenilalanina, prolina, serina, treonina, triptófano, tirosina y valina, respectivamente.

De acuerdo a Isaev (2006), y en concordancia con la aceptada teoría de la evolución, las secuencias de ADN, ARN y proteínas son creadas a partir de secuencias ya existentes mediante algunas mutaciones. Este hecho es el fundamento de cualquier análisis de secuencias, pues si logramos relacionar una secuencia recién descubierta con algunas secuencias sobre las cuales ya se es conocida su estructura o su función, entonces podremos realizar inferencias sobre la estructura o función de la secuencia nueva. De esta forma, establecer métodos para relacionar secuencias es primordial para la biología. Basándonos en el hecho del origen evolutivo antes descrito, se considera a los alineamientos de secuencias como encapsulamientos de la información evolutiva que dio origen a las secuencias, esta información se almacena en forma matricial en un arreglo de secuencias.

Consideremos la secuencia de ADN $seq = AGGTC$. Al replicarse, tres posibles mutaciones que pueden ocurrir a seq son:

- Sustitución: Un nucleótido es intercambiado por otro. Un posible descendiente de seq al suceder esta mutación es $TGGTC$, donde el primer nucleótido A fue sustituido por el nucleótido T .
- Inserción: Un nucleótido es añadido a la secuencia. En el caso de seq , se podría generar la secuencia $CAGGTC$, dónde el nucleótido C fue insertado al inicio de la secuencia.
- Eliminación: Un nucleótido es eliminado de la secuencia. Un posible descendiente es $GGTC$, donde el primer nuclérido fue eliminado de la secuencia.

Por tanto, dos posibles descendientes de seq son $seq_1 = AGCGTC$ (donde se dio una inserción) y $seq_2 = AGGC$ (donde se dio una eliminación). Representaremos con un gap ‘ - ’ a las mutaciones de inserción y eliminación para visualizar las mutaciones anteriores:

$$\begin{aligned} seq : & \quad AG - GTC \\ seq_1 : & \quad AGCGTC \end{aligned} \tag{2.1}$$

$$\begin{aligned} seq : & \quad AGGTC \\ seq_2 : & \quad AGG - C \end{aligned} \tag{2.2}$$

En el alineamiento 2.1, dado que seq es ancestro de seq_1 , es claro que se está representando una sola mutación que es una inserción. Sin embargo, si no se tuviera la seguridad de que secuencia es la ancestro, entonces el alineamiento podría representar que seq fue obtenida a partir de seq_1 mediante una eliminación. De forma similar en el alineamiento 2.2, es claro que se está representando que seq_2 se obtuvo mediante una eliminación en seq . Sin embargo, en general no se contará con toda la historia como en este caso, mas bien se contará con un conjunto de secuencias sobre las cuales se tratará de encontrar un historial evolutivo.

Por ejemplo, al considerar el alineamiento 2.3 sin conocer más información sobre el origen de las secuencias, se pueden asociar los siguientes históricos evolutivos:

- seq_1 es ancestro de seq_2 , en cuyo caso los gaps denotarían eliminaciones.
- seq_2 es ancestro de seq_1 , en este caso los gaps denotan inserciones.
- seq_1 y seq_2 provienen de algún ancestro en común, por tanto los gaps pueden denotar tanto eliminaciones como inserciones según se considere cómo estuvo conformado el ancestro en común.

$$\begin{aligned} seq_1 : & \quad AGCGTC \\ seq_2 : & \quad AG - G - C \end{aligned} \tag{2.3}$$

Es por esto que incorporar información adicional como secuencias ancestros o historial evolutivo es importante para un análisis más exacto.

En el ejemplo anterior podemos notar que existen diferentes alternativas para alinear dos secuencias. Luego, es necesario establecer criterios para medir la calidad de los alineamientos. La manera usual de hacer esto consiste en determinar un score para cada alineamiento y tomar aquel que lo maximice. Dado un alineamiento, es común asumir independencia sobre sus columnas para determinar el score Isaev (2006), de esta manera solo es necesario obtener un score s para el alineamiento de los distintos elementos de \mathbb{Q} y de ellos con un gap. Al score de un elemento de \mathbb{Q} y un gap lo denominaremos gap penalty.

En el ejemplo anterior, ya que $\mathbb{Q} = \{A, C, G, T\}$, tomaremos el score de la siguiente manera $s(a, a) = 1$, $s(a, b) = -1$, $s(a, -) = s(-, a) = -2$ para $a, b \in \mathbb{Q}$. Entonces el score del alineamiento 2.3 es

$$s(A, A) + s(G, G) + s(C, -) + s(G, G) + s(T, -) + s(C, C) = 1 + 1 - 2 + 1 - 2 + 1 = 0.$$

Como mencionan Polanski y Kimmel (2007), se espera que las funciones de score reflejen en alguna medida la verosimilitud de que el alineamiento que estamos propone sea consistente con un modelo propuesto. Consideremos el caso del ADN, un modelo general para las substituciones de los nucleótidos está dado por una cadena de Markov homogénea a tiempo discreto con matriz de transición:

$$P = \begin{pmatrix} p_{AA} & p_{AC} & p_{AG} & p_{AT} \\ p_{CA} & p_{CC} & p_{CG} & p_{CT} \\ p_{GA} & p_{GC} & p_{GG} & p_{GT} \\ p_{TA} & p_{TC} & p_{TG} & p_{TT} \end{pmatrix}. \quad (2.4)$$

En Polanski y Kimmel (2007) se encuentra a detalle como utilizar estas matrices para definir funciones de score. En forma general, se considera la matriz de intensidad Q en la versión continua del modelo, se ajusta Q con algún conjunto de alineamientos de entrada y se consideran las log-verosimilitudes de los posibles emparejamientos de elementos.

Tres de los modelos más usados para parametrizar P de acuerdo a Polanski y Kimmel (2007) son los siguientes:

- Jukes-Cantor: Se asumen probabilidades equivalentes en las sustituciones:

$$P = \begin{pmatrix} 1 - 3\alpha & \alpha & \alpha & \alpha \\ \alpha & 1 - 3\alpha & \alpha & \alpha \\ \alpha & \alpha & 1 - 3\alpha & \alpha \\ \alpha & \alpha & \alpha & 1 - 3\alpha \end{pmatrix}.$$

- Felsenstein: Se asume que las probabilidades de transición son proporcionales (con coeficiente de proporcionalidad u) a las probabilidades de la distribución estacionaria ($\varphi_A, \varphi_C, \varphi_G, \varphi_T$):

$$P = \begin{pmatrix} 1 - u\varphi_{CTG} & u\varphi_C & u\varphi_G & u\varphi_T \\ u\varphi_A & 1 - u\varphi_{AGT} & u\varphi_G & u\varphi_T \\ u\varphi_A & u\varphi_C & 1 - u\varphi_{ACT} & u\varphi_T \\ u\varphi_A & u\varphi_C & u\varphi_G & 1 - u\varphi_{ACG} \end{pmatrix},$$

donde

$$\varphi_{CTG} = \varphi_C + \varphi_T + \varphi_G,$$

$$\varphi_{AGT} = \varphi_A + \varphi_G + \varphi_T,$$

$$\varphi_{ACT} = \varphi_A + \varphi_C + \varphi_T,$$

$$\varphi_{ACG} = \varphi_A + \varphi_C + \varphi_G.$$

- HKY: Este modelo toma en cuenta la distinción entre transiciones y transversiones. Las transiciones son las mutaciones A-G, G-A, C-T, T-C; mientras que las transversiones corresponden a A-C, C-A, A-T, T-A, C-G, G-C, G-T, T-A:

$$P = \begin{pmatrix} 1 - \psi_{CTG} & v\psi_C & u\varphi_G & v\psi_T \\ v\psi_A & 1 - \psi_{AGT} & v\psi_G & u\psi_T \\ u\psi_A & v\psi_C & 1 - \psi_{ACT} & v\psi_T \\ v\psi_A & u\psi_C & v\psi_G & 1 - \psi_{ACG} \end{pmatrix},$$

donde

$$\psi_{CTG} = u\psi_G + v(\psi_C + \psi_T),$$

$$\psi_{AGT} = u\psi_T + v(\psi_A + \psi_G),$$

$$\psi_{ACT} = u\psi_A + v(\psi_C + \psi_T),$$

$$\psi_{ACG} = u\psi_C + v(\psi_A + \psi_G).$$

Para el caso de los aminoácidos, las matrices de score más utilizadas son las matrices PAM y BLOSUM (Pevsner, 2015).

2.1.1. Cantidad de alineamientos

Ahora, veamos cuantos alineamientos posibles existen para dos secuencias. Consideremos las seq_1 , seq_2 anteriores que queremos alinear y denotemos sus longitudes por l_1 y l_2 , respectivamente. Consideremos el tablero en la Figura 2.2:

Es posible asociar cada alineamiento con una sucesión del conjunto de movimientos {flecha diagonal, flecha abajo, flecha derecha}, como se observa en el tablero. En general, esto se puede hacer identificando:

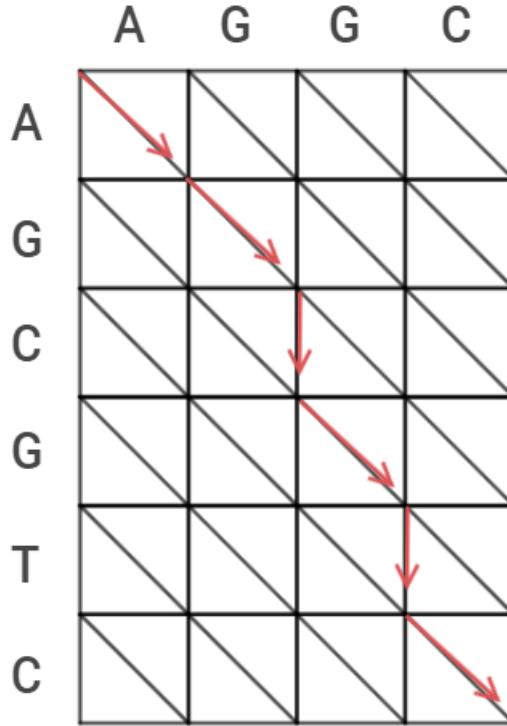


Figura 2.2: Posible camino de la esquina superior izquierda a la esquina inferior derecha. El camino indicado por las flechas refleja el alineamiento 2.1.

- Flecha diagonal: Corresponde a emparejar ambos caracteres asociados en la cuadricula agregandolos a sus respectivas secuencias.
- Flecha abajo: Agregar un gap a s_2 y agregar la letra correspondiente a s_1 .
- Flecha derecha: Agregar un gap a s_1 y agregar la letra correspondiente a s_2 .

Notemos que para producir un alineamiento en total son necesarios l_1 movimientos hacia abajo y l_2 movimientos hacia la derecha. Por tanto, si se realizan k movimientos en diagonal, la cantidad de movimientos hacia abajo restantes son $l_1 - k$ y de movimientos hacia la derecha son $l_2 - k$. Luego, para un k fijo se tiene que la cantidad de movimientos válidos equivale a la cantidad de formas de acomodar $k + l_1 - k + l_2 - k = l_1 + l_2 - k$ flechas de tres tipos distintos, lo cual se puede calcular con el coeficiente multinomial

$$\binom{l_1 + l_2 - k}{k, l_1 - k, l_2 - k} = \frac{(l_1 + l_2 - k)!}{k!(l_1 - k)!(l_2 - k)!}.$$

Como la cantidad de flechas diagonales puede tomar cualquier valor entre $0, \dots, \min(l_1, l_2)$, la cantidad de posibles alineamientos es

$$\sum_{k=0}^{\min(l_1, l_2)} \frac{(l_1 + l_2 - k)!}{k!(l_1 - k)!(l_2 - k)!}.$$

En particular, dicha cantidad está inferiormente acotada por el primer sumando $k = 0$, es decir por

$$\binom{l_1 + l_2}{l_1} = \binom{l_1 + l_2}{l_2}.$$

Luego, la cantidad total de alineamientos crece combinatoricamente según la longitud de las secuencias. De esta forma, para este caso (y para el caso del alineamiento múltiple de secuencias) no es factible revisar todas las posibles opciones.

2.1.2. Algoritmos para alinear dos secuencias

Como se comentó en el Capítulo 1, se distinguen dos tipos de alineamientos que resultan de interés: El alineamiento global y el alineamiento local. En el alineamiento global se busca encontrar una distribución de los gaps de tal forma que el score evaluado sobre las secuencias completas sea máximo; en el alineamiento local se quieren encontrar sub-regiones de las secuencias cuyo score de alineamiento resulte máximo. Procederemos a revisar algunos de los algoritmos usualmente utilizados para obtener alineamientos globales y local de dos secuencias.

Needleman-Wunsch

Utilizando una penalización lineal sobre la aparición de los gaps, esto es, para todo $a \in \mathbb{Q}$, $S(a, -) = S(-, a) = -d$, el algoritmo de Needleman-Wunsch ([Isaev, 2006](#)), encuentra un alineamiento óptimo mediante programación dinámica. Para esto, establezcamos notación que mantendremos a lo largo de este escrito. Sea $seq_1 = seq_1^{(1)} seq_1^{(2)} \dots seq_1^{(l_1)}$ y $seq_2 = seq_2^{(1)} seq_2^{(2)} \dots seq_2^{(l_2)}$ dos secuencias de elementos de \mathbb{Q} con longitud l_1 y l_2 , respectivamente, y tal que $seq_1^{(i)}, seq_2^{(j)} \in \mathbb{Q}$ para $1 \leq i \leq l_1$,

$1 \leq j \leq l_2$. Definimos los prefijos $seq_1^i = seq_1^{(1)}seq_1^{(2)}\dots seq_1^{(i)}$ para $1 \leq i \leq l_1$ y $seq_2^j = seq_2^{(1)}seq_2^{(2)}\dots seq_2^{(j)}$ para $1 \leq j \leq l_2$. Denotaremos por $seq_1^0 = seq_2^0 = \emptyset$ la secuencia vacía. Convengamos que el score de alinear dos cadenas vacías es cero y que el alineamiento de una secuencia vacía con una secuencia no vacía consiste en transformar la secuencia vacía en una secuencia que conste solo de gaps, cuya longitud sea la misma que la otra secuencia en cuestión.

Sea $F(i, j)$ el máximo score que se puede obtener al alinear seq_1^i con seq_2^j . Notemos que existen algunos casos cuyo cálculo es inmediato de las convenciones anteriores:

- $F(0, 0) = 0$, pues representa el alineamiento de dos secuencias vacías;
- $F(i, 0) = -i \times d$ para $1 \leq i \leq l_1$; y
- $F(0, j) = -j \times d$ para $1 \leq j \leq l_2$.

Los casos restantes, podemos calcularlos recursivamente. Si $1 \leq i \leq l_1$ y $1 \leq j \leq l_2$, entonces los últimos elementos de seq_1^i y seq_2^j son $seq_1^{(i)}$ y $seq_2^{(j)}$, respectivamente. Sobre este par de últimos elementos, se puede dar sólamente uno de los siguientes tres casos:

- $seq_1^{(i)}$ y $seq_2^{(j)}$ están emparejados. En este caso, el emparejamiento aporta $S(seq_1^{(i)}, seq_2^{(j)})$ al score, y resta por alinear en forma óptima seq_1^{i-1} con seq_2^{j-1} obteniendo un score $F(i - 1, j - 1)$.
- $seq_1^{(i)}$ está emparejado con un gap añadido en seq_2 . Luego, el emparejamiento aporta $-d$ al score y resta por alinear en forma óptima seq_1^{i-1} con seq_2^j obteniendo un score $F(i - 1, j)$.
- $seq_2^{(j)}$ está emparejado con un gap añadido en seq_1 . En este caso, el emparejamiento aporta $-d$ al score y resta por alinear en forma óptima seq_1^i con seq_2^{j-1} obteniendo un score $F(i, j - 1)$.

F	0	1	2	3	4
0	-	G	T	A	A
1	C	-2	-1	-3	-5
2	T	-4	-3	0	-2
3	T	-6	-5	-2	-1
4	A	-8	-7	-4	0
5	G	-10	-7	-6	-3
6	A	-12	-9	-8	-5

Figura 2.3: Tomando $d = 2$, $S(a, b) = -1$ para $a \neq b$ y $S(a, a) = 1$, se llena la tabla para $seq_1 = CTTAGA$, $seq_2 = GTAA$.

Luego, para $1 \leq i \leq l_1$ y $1 \leq j \leq l_2$, se calcula $F(i, j)$ como

$$F(i, j) = \max \begin{cases} F(i - 1, j - 1) + S(seq_1^{(i)}, seq_2^{(j)}), \\ F(i - 1, j) - d, \\ F(i, j - 1) - d. \end{cases}$$

Una manera de realizar estos cálculos sin utilizar recursividad computacional es creando una matriz de $(l_1 + 1) \times (l_2 + 1)$, llenando los valores $F(0, 0)$, $F(0, j)$, $F(i, 0)$ como se describió anteriormente; calculando los valores restantes de arriba a abajo (i creciendo) y de izquierda a derecha (j creciendo), guardando el dato de cuales opciones de emparejamiento dieron el máximo score. Un ejemplo tomado de [Isaev \(2006\)](#) se puede ver en la Figura 2.3.

Sobre la complejidad del algoritmo anterior, podemos notar que el llenado de cada celda de la matriz toma una cantidad constante de operaciones, luego la complejidad

en tiempo es $O(l_1 \times l_2)$. Considerando $l = l_1 \approx l_2$, la complejidad del algoritmo queda $O(l^2)$, es decir, es un algoritmo cuadrático. Cuando se requiere alinear múltiples secuencias, o alinear una gran cantidad de parejas de secuencias, suelen usarse heurísticas en vez del método exacto, pues dependiendo de la cantidad de alineamientos por realizar y la longitud promedio de las secuencias, puede resultar poco factible utilizar siempre este algoritmo.

Smith-Waterman

El algoritmo de Smith-Waterman ([Isaev, 2006](#)) obtiene mediante programación dinámica el alineamiento local (es decir, entre segmentos consecutivos de las secuencias en cuestión) con score máximo. La forma de proceder es muy similar al algoritmo de Needleman-Wunsch, pero con un cambio en la fórmula recursiva:

$$F(i, j) = \max \begin{cases} 0, \\ F(i - 1, j - 1) + S(\text{seq}_1^{(i)}, \text{seq}_2^{(j)}), \\ F(i - 1, j) - d, \\ F(i, j - 1) - d \end{cases}$$

La opción adicional del cero nos dice que si el alineamiento óptimo de seq_1^i y seq_2^j tiene score negativo, entonces es mejor ignorar estos prefijos en los casos subsecuentes que de este se desprenden considerándolo como si se hubiesen alineado dos secuencias vacías. Un ejemplo tomado de [Isaev \(2006\)](#) de este algoritmo se puede ver en la Figura 2.4. En este caso, para reconstruir una solución se encuentra primero el máximo valor en la matriz y se rastrea el camino desde ese punto.

FASTA y BLAST

Dos algoritmos usualmente utilizados como alternativas a los métodos exactos son los heurísticos FASTA y BLAST. FASTA ([Polanski y Kimmel, 2007](#)) aplica el algoritmo de la programación dinámica sobre una reducción del conjunto de estados; para esto, se encuentran regiones de similaridad basadas en un k -mero. Por ejemplo en el caso descrito en la Figura 2.5, que fue tomado de [Isaev \(2006\)](#), se sombrean regio-

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

Figura 2.4: Tomando $d = 2$, $S(a, b) = -1$ para $a \neq b$ y $S(a, a) = 1$, se llena la tabla para $seq_1 = CTTAGA$, $seq_2 = GTAA$.

nes de similaridad dadas por 3-meros. Una vez localizadas dichas regiones, se toman bandas alrededor de estas diagonales como el espacio factible de búsqueda, como se puede ver en la Figura 2.6. Finalmente se procede mediante alguno de los métodos de programación dinámica sobre el espacio reducido. Notemos que esta reducción ocasiona que no se pueda garantizar la obtención de un alineamiento con score máximo, pero es de esperarse que el alineamiento obtenido sea de buena calidad.

BLAST es un algoritmo puramente heurístico que no está basado en programación dinámica como FASTA. En primera instancia, BLAST localiza regiones de similaridad, como se hace en FASTA, para después extender dichas regiones mediante criterios heurísticos basados en la verosimilitud de que la extensión de las regiones mantengan cierto grado de similaridad (Polanski y Kimmel, 2007).

2.2. Alineamiento múltiple de secuencias

Es de interés también analizar conjuntos de más de dos secuencias. La idea antes utilizada para encapsular la información de la evolución que dio origen a dos secuencias

Dot Matrix	G	T	C	A	G	A	C	G	C	T	C	A
C			*				*		*		*	
A				*		*						*
G	*				*			*				
A				*		*						*
G	*				*			*				
T		*								*		
T		*								*		
A				*		*						*
C			*				*		*		*	
G	*				*			*				
T		*								*		
C			*				*		*		*	
A				*		*						*

Figura 2.5: Matriz donde se localizan 3–meros dadas dos secuencias.

F	G	T	C	A	G	A	C	G	C	T	C	A
C					-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
A						-∞	-∞	-∞	-∞	-∞	-∞	-∞
G							-∞	-∞	-∞	-∞	-∞	-∞
A								-∞	-∞	-∞	-∞	-∞
G	-∞								-∞	-∞	-∞	-∞
T	-∞	-∞								-∞	-∞	-∞
T	-∞	-∞	-∞								-∞	-∞
A	-∞	-∞	-∞	-∞								-∞
C		-∞	-∞	-∞	-∞							
G			-∞	-∞	-∞	-∞						
T				-∞	-∞	-∞	-∞					
C	-∞				-∞	-∞	-∞	-∞				
A	-∞	-∞				-∞						

Figura 2.6: Región de búsqueda acotada mediante el algoritmo FASTA, tomando las diagonales vecinas de las regiones previamente identificadas.

de entrada, se extiende naturalmente al caso de múltiples secuencias. Consideremos $N > 2$ secuencias $seq_1, seq_2, \dots, seq_N$. En este caso, se agregan gaps a las N secuencias de tal manera que al final todas ellas tengan la misma longitud L generando una matriz $A = (a_i^{(j)})$ de $N \times L$. Para medir la calidad de los alineamientos también se utilizan scores. El score que suele utilizarse para medir la calidad de alineamientos múltiples es el SP-score, que se forma considerando todas las parejas de elementos por cada columna y realizando la suma de los scores dados por S como en la sección anterior. Es decir, se tiene que el SP-score del alineamiento A está dado por

$$SP(A) = \sum_{c=1}^L \sum_{i=1}^{N-1} \sum_{j=i+1}^N S(a_i^{(c)}, a_j^{(c)}),$$

donde a_i representa la i -ésima fila de A y $a_i^{(c)}$ representa el c -ésimo elemento de la i -ésima fila de A . De esta forma, si \mathbb{A} representa el conjunto de todos los posibles alineamientos de las secuencias seq_1, \dots, seq_N , el *problema del alineamiento múltiple de secuencias* de las secuencias seq_1, \dots, seq_N corresponde a encontrar un $A^* \in \mathbb{A}$ tal que $SP(A) \leq SP(A^*)$ para todo $A \in \mathbb{A}$. Observemos que si bien la cantidad total de alineamientos es muy grande, este es un número finito, y por tanto siempre existe cuando menos un $A^* \in \mathbb{A}$ que satisfaga la condición requerida.

Es posible generalizar la estrategia de la programación dinámica al caso de los alineamiento múltiples. Considerando la notación en forma análoga a lo antes establecido, sea $F(i_1, i_2, \dots, i_N)$ el score máximo que se puede obtener al alinear $seq_1^{i_1}, seq_2^{i_2}, \dots, seq_N^{i_N}$. En este caso también es posible establecer una relación recursiva, solo que en esta ocasión, cuando estamos considerando los sub casos asociados tendremos que elegir en cuales de las N secuencias colocar un gap, esto implica tomar N decisiones binarias. Por tanto descartando la opción de colocar un gap en todas las secuencias, la cantidad de casos son $2^N - 1$. Luego, la complejidad en este caso es $O(l_1 \times l_2 \dots l_N \times 2^N)$; si L es la longitud promedio de las secuencias, se tiene que la complejidad del algoritmo en este caso es de $O(L^N \times 2^N)$. Considerar este algoritmo es claramente inviable en la mayoría de los casos.

-	A	G	G	C	T	A	T	C	A	C	C	T	G
T	A	G	-	C	T	A	C	C	A	-	-	-	G
C	A	G	-	C	T	A	C	C	A	-	-	-	G
C	A	G	-	C	T	A	T	C	A	C	-	G	G
C	A	G	-	C	T	A	T	C	G	C	-	G	G
A		1			1		.8						
C	.6			1		.4	1	.6	.2				
G			1	.2				.2		.4	1		
T	.2				1	.6				.2			
-	.2		.8					.4	.8	.4			

Figura 2.7: Perfil resumen de un alineamiento de 5 secuencias, en cada columna se coloca la proporción de elementos para cada elemento de \mathbb{Q} .

Debido a la observación anterior, este problema es generalmente abordado mediante heurísticas que aportan en la calidad de alineamientos, o que generan un ahorro de coste computacional. En esta línea, dos de los principales métodos usados para realizar alineamiento múltiple de secuencias son CLUSTALW y MAFFT. Ambos enfoques, más que un solo algoritmo, ofrecen una gamma de modelos que se puede personalizar mediante una interacción con el usuario. Ambos métodos están basados en el alineamiento progresivo de secuencias; dicho procedimiento consiste en realizar múltiples alineamientos de parejas de perfiles asociados a alineamientos hasta obtener un solo alineamiento que involucre a todas las secuencias. En un perfil asociado a un alineamiento, cada alineamiento es transformado a una secuencia de vectores ($|\mathbb{Q}| + 1$)-dimensionales donde cada vector corresponde a una columna del alineamiento y representa las proporciones de aparición de los elementos de \mathbb{Q} y el gap en dicha columna. Por ejemplo, si una columna de un alineamiento es *ATTGG — G*, esta información es resumida en un vector de 5 coordenadas $(\frac{1}{8}, \frac{2}{8}, \frac{3}{8}, \frac{0}{8}, \frac{2}{8})$, donde las entradas representan la proporción de *A*, *G*, *T*, *C*, — presentes en la columna, respectivamente. En la Figura 2.7 se encuentra otro ejemplo de un perfil. El alineamiento de perfiles puede realizarse extendiendo el algoritmo Needlema-Wunsch, cada perfil es considerado como una sola secuencia y se toma como score entre elementos de los perfiles a el producto punto ponderado por la matriz de score S de los respectivos vectores en cuestión.

En las siguientes subsecciones se explican de manera general algunos de los algo-

ritmos más utilizados actualmente para el alineamiento múltiple de secuencias.

2.2.1. CLUSTALW

CLUSTALW es la tercera generación del algoritmo CLUSTAL de Higgins y Sharp (1988), que según lo encontrado en la wikipedia, su nombre hace referencia al *CLUSTer anAlysis* que se realiza durante la ejecución del método. Como se explica en Pevsner (2015), este algoritmo se compone de tres fases:

1. Realizar el alineamiento de los $\binom{N}{2}$ pares de secuencias, con la finalidad de obtener una matriz de distancias utilizando los scores de cada alineamiento.
2. En base a la matriz de distancias, se crea un árbol guía. Usualmente se utiliza el algoritmo UPGMA (unweighted pair group method with arithmetic mean) (Pevsner, 2015) o el algoritmo Neighbor-Joining (Saitou y Nei, 1987).
3. Una vez conseguido el árbol guía, se alinean las secuencias, según lo indica el árbol, mediante la técnica de perfiles antes mencionada.

El ejemplo mostrado en la Figura 2.8, tomado de Pevsner (2015), ilustra la construcción del árbol guía. En este contexto, la tercera fase consiste en:

1. Alinear las secuencias s_1 y s_2 para obtener el perfil p_{12} .
2. Alinear las secuencias s_4 y s_5 para obtener el perfil p_{45} .
3. Alinear p_{45} y s_3 para obtener el perfil p_{453} .
4. Alinear p_{12} con p_{453} para obtener el alineamiento final.

2.2.2. MAFFT

El algoritmo MAFFT (multiple alignment using fast Fourier transform) tiene tres variantes que describiremos a continuación de forma general, los detalles se pueden encontrar en (Katoh, Misawa, Kuma, y Miyata, 2002):

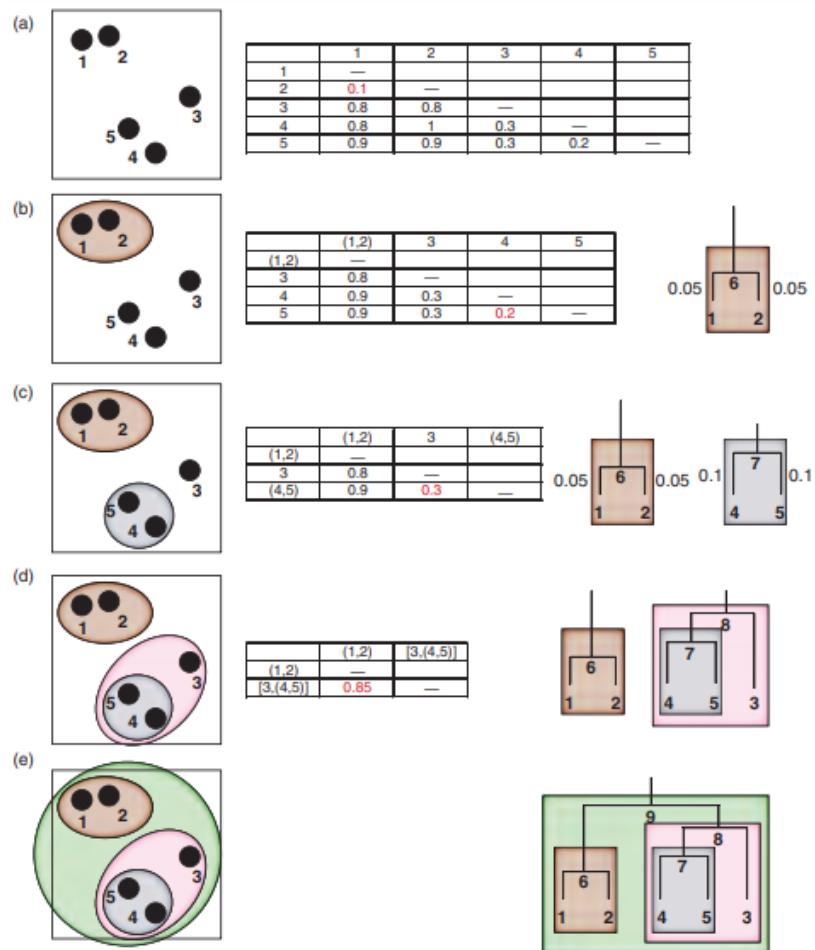


Figura 2.8: Agrupamiento de 5 secuencias dadas una matriz de distancias.

- **FFT-NS-1:** Es similar a CLUSTALW con respecto a las primeras dos fases. El cambio está en que el alineamiento entre perfiles/secuencias se realiza con una variante del algoritmo de Needleman-Wunch que en vez de tener una complejidad de $O(L^2)$, tiene complejidad $O(L \times \log L)$.
- **FFT-NS-2:** Usando el alineamiento obtenido por FFT-NS-1, se crea un nuevo árbol guía y se vuelve a realizar los correspondientes alineamientos.
- **FFT-NS-i:** Mediante un proceso iterativo, se partitionan en dos grupos las secuencias para re-alinear los perfiles correspondientes.

La variante del algoritmo de Needleman-Wunsch consta de los siguientes pasos:

- Obtener regiones de las secuencias que sean potencialmente similares. Para esto, se calculan correlaciones traslaciones de las cadenas utilizando el algoritmo de la Transformada Rápida de Fourier (FFT, por sus siglas en inglés). La Figura 2.9 ilustra este proceso de forma gráfica.
- Elegir un apareamiento de regiones similares que maximicen el score de emparejamiento mediante programación dinámica.
- Con el emparejamiento de regiones, se realiza una reducción del espacio de búsqueda de la programación dinámica sobre las secuencias completas y se ejecuta el algoritmo de Needleman-Wunch sobre el espacio reducido. Ver la Figura 2.10.

2.2.3. MUSCLE

El siguiente método a considerar es MUSCLE (MULTiple Sequence Comparison by Log-Expectation) [Edgar \(2004b\)](#). Este algoritmo consta de tres etapas:

1. En la primera fase, se procede en forma similar a CLUSTALW. La diferencia es que la matriz de distancias se construye realizando conteo de k -meros (cadenas de caracteres de longitud k); esto se basa en la idea que secuencias más similares compartirán una mayor cantidad de k -meros que las que compartirían por simple

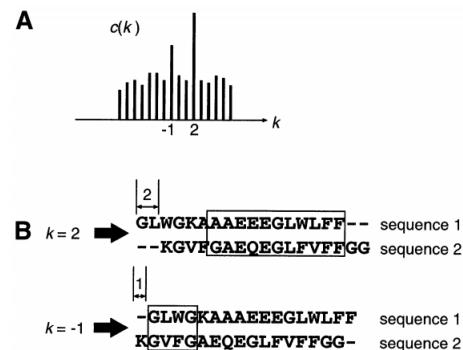


Figura 2.9: En A se obtienen los distintos niveles de correlación de las secuencias trasladadas usando la FFT. En B mediante un algoritmo de ventana deslizante se obtienen regiones de similitud para considerar en la reducción de espacio de búsqueda para la programación dinámica.

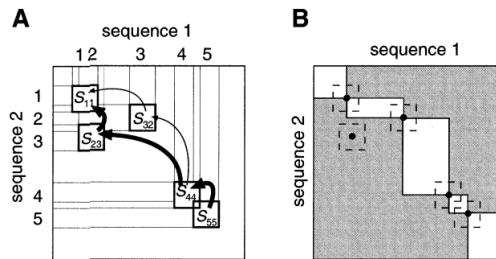


Figura 2.10: En A se elige que regiones localizadas en la parte anterior se quieren fijar para la reducción del espacio de búsqueda presente en B.

aleatoriedad. Una forma de realizar este conteo es usando una Hash-Table de los k -meros que se requieran considerar y hacer un barrido lineal sobre las secuencias en cuestión, en [Edgar \(2004a\)](#) se puede encontrar mas información al respecto. Esto da como resultado un primer alineamiento denominado MSA1.

2. El porcentaje de identidad entre dos secuencias alineadas es la proporción de columnas en las cuales ambas secuencias contienen el mismo nucleótido. En este paso, se obtiene una segunda matriz de distancias calculando los porcentajes de identidad entre todas las parejas de secuencias según MSA1. Esta matriz sirve para calcular un nuevo árbol guía y posteriormente realizar un nuevo alineamiento progresivo para obtener MSA2.
3. Este paso es iterativo, en cada iteración se remueve una arista del árbol calculado en el segundo paso, se calculan los perfiles asociados a los subárboles formados y se realinean dichos perfiles para generar un nuevo alineamiento MSA3. Si el SP-score de MSA3 es mayor que el SP-score del mejor alineamiento hasta el momento, se toma MSA3 como el nuevo mejor alineamiento y se repite el paso 3. En caso contrario, se da por terminado el algoritmo y se reporta el mejor alineamiento observado.

En cada alineamiento entre perfiles realizado, se utiliza una evaluación propuesta por [Edgar \(2004b\)](#) que asigna un score para cada pareja de columnas de alineamientos múltiples basándose en las probabilidades de transición de la matriz PAM 240; este score entre columnas de alineamiento múltiples es denominado log-expectation. En la Figura 2.11 podemos observar un diagrama tomado de [Edgar \(2004b\)](#) con los pasos anteriormente descritos.

2.2.4. T-Coffee

Finalmente, consideremos el algoritmo T-COFFEE (tree-based consistency objective function for alignment evaluation) [Notredame, Higgins, y Heringa \(2000\)](#). La idea general del algoritmo es realizar una versión pesada de CLUSTALW, donde los pesos de los posibles emparejamientos de caracteres reflejen consistencia con respecto a

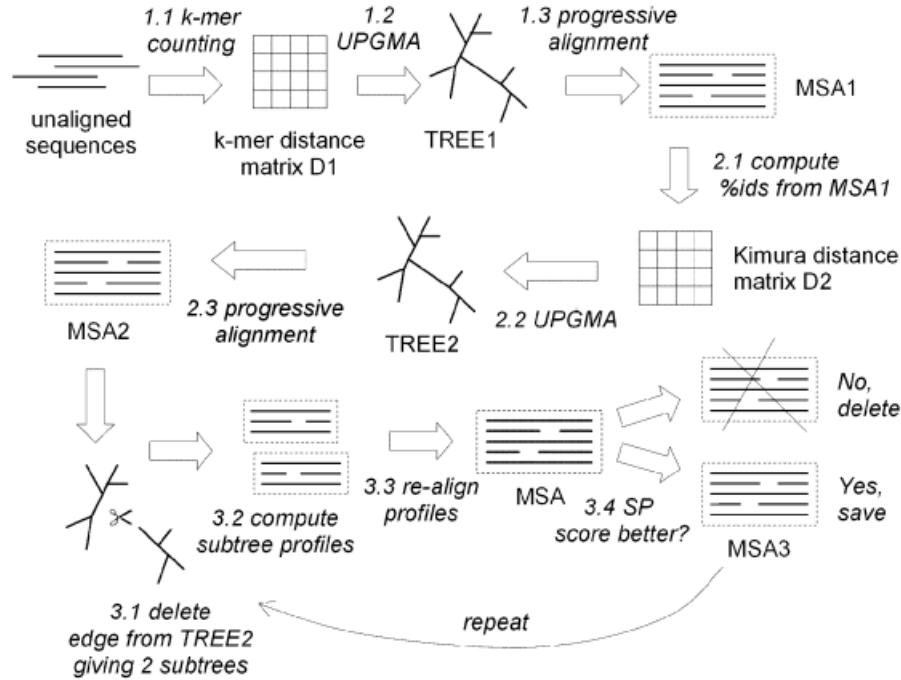


Figura 2.11: Representación de las tres etapas de MUSCLE con sus pasos intermedio.

todo el conjunto de secuencias. Este algoritmo tiene dos etapas. En la primera etapa se tiene como objetivo obtener pesos que se asignaran al alineamiento entre cualesquiera dos elementos de las secuencias a alinear. Este proceso se realiza mediante los siguientes pasos:

1. Se realizan alineamientos globales y locales entre todas las parejas de secuencias, de tal forma que para cada par de secuencias se tiene cuando menos un ejemplo de alineamiento global y posiblemente múltiples ejemplos de alineamientos locales.
2. Considerando todos los alineamientos generados en el paso anterior, se le asigna un peso equivalente a su porcentaje de identidad a todos los elementos de las secuencias que resultaron emparejados en una misma columna. Si una pareja aparece en más de un alineamiento, se le asigna la suma de los pesos de los alineamientos en los que estuvo presente. Si una pareja en particular no aparece, se considera que tiene un peso de cero. Denotamos por W a la función que asigna los pesos antes definidos. Esta asociación de pesos conforma la librería primaria.

3. El siguiente paso consiste en extender la librería para considerar consistencia entre todas las secuencias. Si el i -ésimo elemento de la secuencia seq_x es alineado con el j -ésimo elemento de la secuencia seq_y en al menos un ejemplo de los alineamientos del paso 1, entonces se consideran todas las secuencias seq_z con $z \neq x$ y $z \neq y$ para las cuales existen ejemplos en los alineamientos del paso 1 tal que existe un elemento seq_z^k que esta alineado con seq_x^i y con seq_y^j . Siendo así, al peso $W(seq_x^i, seq_y^j)$ en la librería extendida se le añade $\min(W(seq_x^i, seq_z^k), W(seq_y^j, seq_z^k))$.

Una vez obtenida la librería extendida, se procede a ejecutar la segunda etapa, que consiste en realizar el alineamiento progresivo siguiendo los primeros dos pasos de CLUSTALW, con la diferencia de que en el tercer paso para realizar los alineamientos siguiendo el árbol guía obtenido, se consideran los pesos de la librería extendida en vez de los dictados por la función de score S . En la Figura 2.12 podemos observar un diagrama tomado de [Notredame y cols. \(2000\)](#) con los pasos anteriormente descritos.

2.3. Aprendizaje reforzado

Como podemos encontrar en ([Wiering y Van Otterlo, 2012](#)), el aprendizaje reforzado es una clase de algoritmos dentro del campo del aprendizaje máquina que permiten a un agente aprender como comportarse en un entorno de tal forma que se maximice una señal de recompensa escalar con el paso del tiempo.

Consideremos un proceso de decisión de Markov que es determinado por la tupla (S, A, T, R) , donde S es un conjunto finito de estados; A es un conjunto finito de acciones; T es una función $T : S \times A \times S \mapsto \mathbb{R}$ donde $T(s, a, s')$ indica la probabilidad de transitar al estado s' al realizar la acción a desde el estado s ; y R es una función de recompensa $R : S \times A \times S \mapsto \mathbb{R}$ donde $R(s, a, s')$ indica la recompensa asignada al transitar al estado s' desde el estado s al realizar la acción a . Sobre S se distinguen dos tipos de estados en particular, el conjunto de estados iniciales I y finales F . Suele representarse $A(s)$ como el subconjunto de acciones que es válido tomar desde el es-

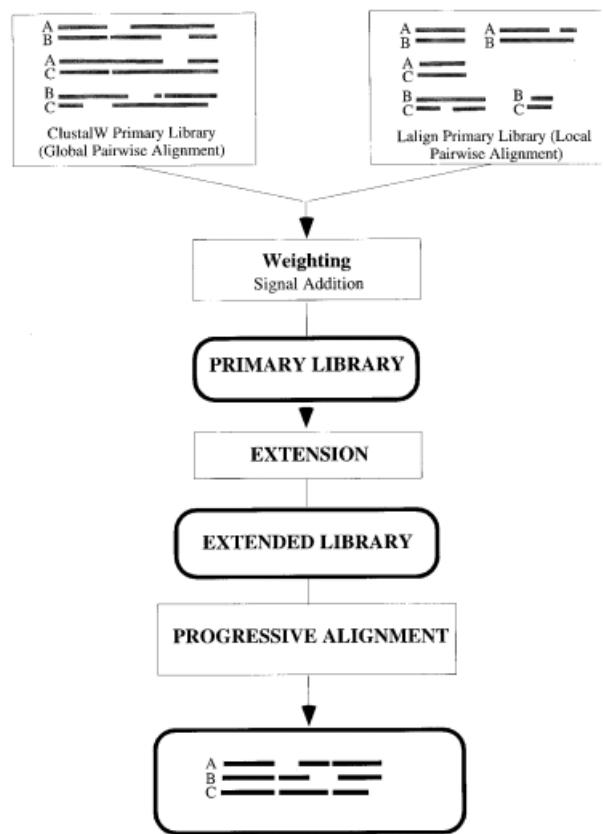


Figura 2.12: Representación de las distintas etapas del algoritmo T-COFFEE.

tado s , esto refleja que no todas las acciones se pueden realizar sobre todos los estados.

Una política determinista π es una función $\pi : S \mapsto A$ que a cada estado determina que acción tomar. Aplicar una política desde un estado inicial s_0 genera una secuencia de estados-acciones-recompensas dada por $s_0, a_0, r_0, s_1, a_1, r_1$, etc. Dada $0 < \gamma < 1$, se define V como la función de valor del estado que asigna el valor esperado de la suma descontada por γ de las recompensas observadas al seguir la política π como

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_k | s_0 = s \right\}.$$

Notemos que debido a que la indexación nunca afecta a la aplicación de las funciones T , π , R , entonces la definición anterior es invariante bajo la numeración del primer subíndice. Es decir, para todo $t \geq 0$

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\}. \quad (2.5)$$

Similarmente se define la función de valor del estado-acción

$$Q^\pi(s, a) = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a \right\}. \quad (2.6)$$

Aquí $Q^\pi(s, a)$ representa el valor esperado de la suma descontada por γ de las recompensas observadas al tomar la acción a desde el estado s y posteriormente seguir la política π .

Se define una política óptima π^* como una política tal que para todo $s \in S$ y toda política π se tiene que $V^{\pi^*}(s) \geq V^\pi(s)$. Resolver un proceso de decisión de Markov consiste en encontrar una política óptima y el aprendizaje reforzado es una estrategia que se puede emplear con dicho fin.

En las aplicaciones que se revisarán en este trabajo de tesis, se cumple que las

transiciones son deterministas, es decir se tiene que para todo $s \notin F$, $a \in A(s)$

$$T(s, a, s') \in \{0, 1\} \quad \forall s' \in S,$$

y solo un único $s' \in S$ cumple $T(s, a, s') = 1$. De esta manera, de ahora en adelante se considera a la función $T : S \times A \mapsto S$ como la función que asigna a cada par válido estado-acción su correspondiente estado destino. Similarmente, se considera ahora la notación $R : S \times A \mapsto \mathbb{R}$ como la recompensa correspondiente a elegir la acción a desde el estado s . Por tanto, podemos reescribir $s_0, a_0, r_0, s_1, a_1, r_1$, etc. como $a_0 = \pi(s_0)$, $r_0 = R(s_0, a_0)$, $s_1 = T(s_0, a_0)$, $a_1 = \pi_{s_1}$, $r_1 = R(s_1, a_1)$, $s_2 = T(s_1, a_1)$, etc.

De esta forma, para una política determinista fija π y un estado s se tiene que $s_0, a_0, r_0, s_1, a_1, r_1$, es una secuencia de estados-acciones-recompensas única, por tanto las esperanzas presentes en las ecuaciones (2.5) y (2.6) se convierte en evaluaciones de la siguiente manera:

$$V^\pi(s) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big|_{s_t=s}, \quad (2.7)$$

$$Q^\pi(s, a) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big|_{s_t=s, a_t=a}. \quad (2.8)$$

De lo anterior, notemos que

$$\begin{aligned} V^\pi(s) &= \sum_{k=0}^{\infty} \gamma^k r_{t+k} \Big|_{s_t=s} = r_t + \gamma \left(\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \Big|_{s_{t+1}=T(s, \pi(s))} \right) \\ &= R(s, \pi(s)) + \gamma V^\pi(T(s, \pi(s))). \end{aligned}$$

Es fácil observar (mediante contradicción) que $V^* = V^{\pi^*}$ satisface la ecuación de Bellman

$$V^*(s) = \max_{a \in A(s)} [R(s, a) + \gamma V^*(T(s, a))].$$

Más aún, dada V podemos construir una política-greedy

$$\pi_{greedy}(V)(s) = \arg \max_{a \in A(s)} [R(s, a) + \gamma V(T(s, a))]$$

donde se cumple $V^* = V^{\pi_{greedy}(V^*)}$. Similarmente, dada una política óptima, su función del valor estado-acción cumple

$$Q^*(s, a) = R(s, a) + \gamma \max_{a' \in A(T(s, a))} Q^*(T(s, a), a'), \quad (2.9)$$

$$\pi^*(s) = \arg \max_{a \in A(s)} Q^*(s, a). \quad (2.10)$$

Además, se tiene la igualdad $V^*(s) = \max_{a \in A} Q^*(s, a)$. De esta forma, una estrategia para encontrar una política óptima es mediante la búsqueda de una función de estado-valor Q que satisfaga la relación (2.9). Una estrategia para lograr esto es mediante la inicialización aleatoria de la función Q (o bien, asumir que inicialmente $Q(s, a) = 0 \quad \forall s \in S, a \in A$), para posteriormente, con la realización de algunos episodios, hacer actualizaciones del valor de la Q para acercarnos hacia el cumplimiento de (2.9); este algoritmo es llamado Q -learning (ver Algoritmo 1). En dicho algoritmo α representa en el caso más general la tasa de aprendizaje que representa la confiabilidad que le asignamos a la nueva predicción $r + \gamma \max_{a' \in A(s')} Q(s', a')$ con respecto al valor anterior de $Q(s, a)$. Dos estrategias de exploración en la generación de episodios son las siguientes:

- ε -greedy: Dado un $0 \leq \varepsilon \leq 1$, con probabilidad ε se toma cualquier acción disponible aleatoriamente, con probabilidad $1 - \varepsilon$ se toma la acción con máximo Q valor desde el estado actual. La idea detrás de esta estrategia es buscar un balance entre el descubrimiento cuando elegimos una acción al azar y utilizar la información que el estado actual de Q nos arroja.
- Boltzman (softmax): Dado un estado, se definen las probabilidades para elegir la acción a como

$$P(a) = \frac{e^{\frac{Q(s, a)}{K}}}{\sum_i e^{\frac{Q(s, a_i)}{K}}}$$

donde valores grandes de K tienden a una elección completamente aleatoria, y valores pequeños a una estrategia greedy. Esta estrategia siempre utiliza el

estado actual de Q en la elección de la acción a tomar pues se les asigna probabilidad de elección a todas las acciones disponibles con base a los valores de Q (dando una mayor probabilidad a aquellas acciones con Q valores mayores).

Algoritmo 1 Q -learning.

Require: γ , learning parameter α

Iniciarizar Q arbitrariamente.

for each episodio **do**

Elegir un estado inicial $s \in I$.

repeat

Eligir una acción $a \in A(s)$ basándose en Q y en una estrategia de exploración.

Realizar acción a

Observar el nuevo estado $s' = T(s, a)$ y la recompensa $r = R(s, a)$ obtenida.

$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a' \in A(s')} Q(s', a') - Q(s, a))$

$s := s'$

until $s \in F$

end for

2.3.1. Deep Q-Learning

La manera usual de implementar el algoritmo anterior es mediante el almacenamiento de los valores de Q en una tabla. Notemos que esto puede ser inviable dependiendo de la cantidad de estados presentes. De esta forma surge el Deep Q -learning [Mnih y cols. \(2013\)](#) donde se utilizan redes neuronales como aproximadores de la función Q (ver Figura 2.13). Sin embargo, se necesita tener en consideración nuevas técnicas, pues en cada momento el target de actualización va cambiando y por tanto suelen usarse las estrategias de una red neuronal auxiliar (Figura 2.14) y el experience replay (Figura 2.15).

La versión del experience replay que utilizaremos está basado en el algoritmo utilizado por [Mnih y cols. \(2013\)](#) el cual está mostrado en el algoritmo 2.

2.3.2. Neural Fitted Q Iteration

Otro algoritmo de interés es el Neural Fitted Q Iteration (NFQ) de [Lange, Gabel, y Riedmiller \(2012\)](#). Este algoritmo es útil cuando es posible obtener mediante

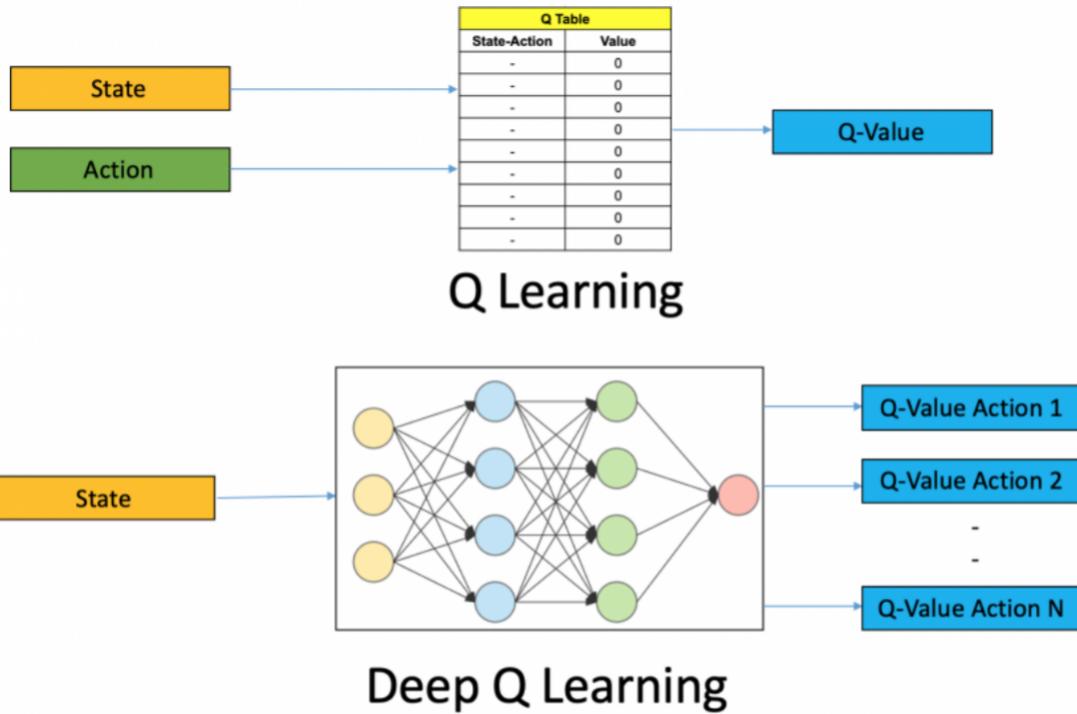


Figura 2.13: Cuando la cantidad de estados es muy grande, entonces se usan redes neuronales para aproximar a la función Q . Figura de Choudhary (2020).

Algoritmo 2 Deep Q -learning with Experience Replay

Require: γ , parámetro de aprendizaje α

Iniciar red neuronal $Q(\cdot; \theta)$ arbitrariamente. θ representa los pesos de la red
Crear repositorio D de experiencias (s, a, r)

for each episodio **do**

Elegir un estado inicial $s \in I$

repeat

Eligir una acción $a \in A(s)$ basándose en Q y en una estrategia de exploración

Realizar acción a

Observar el nuevo estado $s' = T(s, a)$ y la recompensa $r = R(s, a)$ obtenida

Guardar la experiencia (s, a, r) en D

Obtener mini-batch de experiencias de D , conformado por B experiencias elegidas aleatoriamente $(s_j, a_j, r_j) \quad 1 \leq j \leq B$

Definir $y_j := \begin{cases} r_j & T(s_j, a_j) \in F \\ r_j + \gamma \max_{a' \in A(T(s_j, a_j))} Q(T(s_j, a_j), a'; \theta) & T(s_j, a_j) \notin F \end{cases}$

Considerando el parámetro de aprendizaje $y_j := \alpha y_j + (1 - \alpha)Q(s_j, a_j; \theta)$

Realizar descenso por gradiente sobre $(y_j - Q(s_j, a_j; \theta))^2$ para actualizar θ

$s := s'$

until $s \in F$

end for

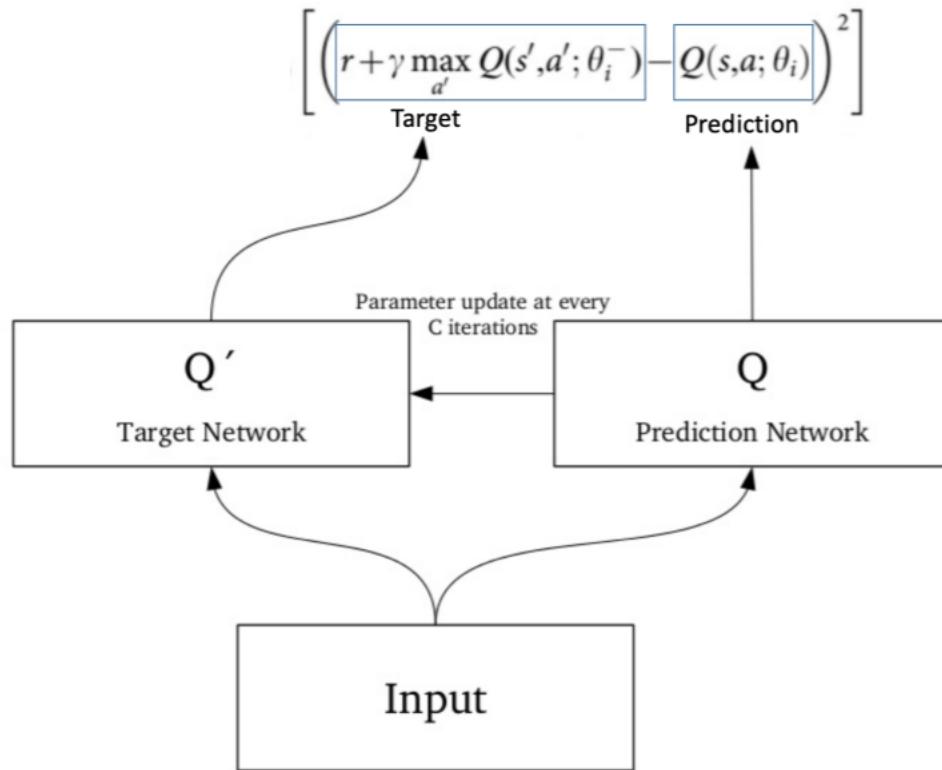


Figura 2.14: Para aminorar los efectos de las actualizaciones de gradiente sobre objetivos móviles, se utiliza una red neuronal auxiliar, y esta se actualiza hasta después de algunas iteraciones. Figura de Choudhary (2020)

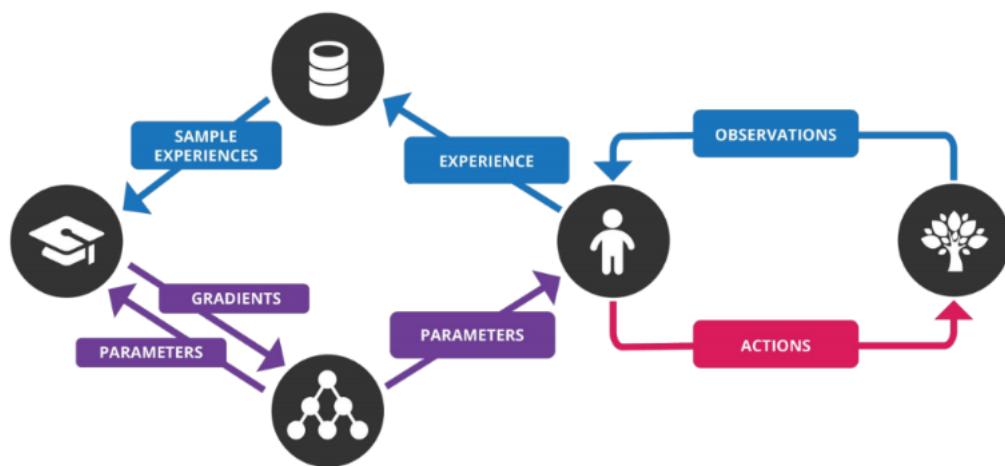


Figura 2.15: Mediante el experience replay, se almacenan las situaciones previamente experimentadas para posteriormente muestrearse con la intención de actualizar la red neuronal. Figura de Choudhary (2020).

exploración o simulación un batch grande de experiencias (s, a, r, s') . Utilizando el batch de experiencias ajustamos iterativamente una red neuronal Q que aproxime a la verdadera función de valor (ver Algoritmo 3).

Algoritmo 3 Neural Fitted Q Iteration

Require: Batch de experiencias $\mathbb{F} = \{(s_t, a_t, r_t, s'_t)\}$ con $1 \leq t \leq B$. N cantidad de iteraciones a realizar.

Iniciar red neuronal \hat{Q}^0 arbitrariamente.

$i := 0$

repeat

Generar conjunto de patrones $P = \{input_t, target_t\}$ con:

$input_t = s_t, a_t$

$target_t = r_t + \gamma \max_{a' \in A} \hat{Q}^i(s'_t, a')$

para $1 \leq t \leq D$

Añadir a P patrones con estados adyacentes a estados finales.

Normalizar y escalar los elementos de P .

Ajustar \hat{Q}^i con los patrones en P para obtener \hat{Q}^{i+1}

$i := i + 1$

until $i = N$

2.4. Enfoques existentes de RL para MSA

Recientemente, [Mircea y cols. \(2014\)](#), [Mircea y cols. \(2016\)](#), [Ramakrishnan \(2018\)](#), [Ramakrishnan y cols. \(2018\)](#) y [Jafari y cols. \(2019\)](#) han abordado el problema del MSA utilizando RL. Entre estos trabajos, los resultados en [Ramakrishnan \(2018\)](#) y [Ramakrishnan y cols. \(2018\)](#) son los menos concluyentes en su aplicación, pues solo se realizaron alineamientos de tres secuencias con un alto coste computacional. Debido a esto nos enfocaremos en los otros tres trabajos.

En [Mircea y cols. \(2014\)](#), [Mircea y cols. \(2016\)](#), [Jafari y cols. \(2019\)](#) se considera el enfoque del alineamiento progresivo restringido a permutaciones de las secuencias (ver Figura 2.16). Esto define el siguiente paradigma estado-acción-recompensa:

- **Estados:** Conjuntos ordenados de índices del 1 al N y tamaño $1 \leq k \leq N$. La cantidad de estados con k elementos es N^k y solo se consideran estados de hasta

N elementos. Así la cantidad total de estados es

$$1 + N + N^2 + \cdots + N^N = \frac{N^{N+1} - 1}{N - 1}.$$

El conjunto con cero elementos es considerado como el único estado inicial, y todos los conjuntos con N elementos definen el conjunto de estados finales.

Consideremos que se requiere alinear las $N = 4$ secuencias $seq_1, seq_2, seq_3, seq_4$; algunos estados válidos son:

- (3) esto representa elegir seq_3 para iniciar con el alineamiento progresivo.
- (2, 2) representa elegir seq_2 y posteriormente volver a elegir seq_2 . A pesar de ser un estado válido, este estado no representa un proceso válido en la ejecución del alineamiento progresivo.
- (2, 1, 3, 4) este estado representa que se ejecutó un alineamiento progresivo de la siguiente forma:
 1. Se alinea seq_2 con seq_1 para obtener el alineamiento p_{21} .
 2. Se alinea p_{21} con seq_3 para obtener el alineamiento p_{213} .
 3. Finalmente, se alinea p_{213} con seq_4 para obtener el alineamiento p_{2134} .

■ **Acciones:** Para cada estado no final s , se consideran las acciones de concatenar al final de s cualquier índice $1 \leq i \leq N$ para llegar al estado $[s, i]$, por tanto se tienen N posibles acciones. (ver Figura 2.17).

■ **Recompensas:** Para estados que no representen el prefijo de una permutación, la recompensa asociada será de $-\infty$. Cuando desde un estado válido s , que tiene asociado el alineamiento y el perfil producido por las secuencias cuyos índices están en s , se accede a otro estado válido $[s, i]$ y se realiza el alineamiento del perfil de s con la i -ésima secuencia. Se toma como recompensa el SP-score restringido a la secuencia i contra el resto del alineamiento obtenido.

Mircea y cols. (2014) y Mircea y cols. (2016) utilizan un software previamente creado por algunos de los autores Czibula, Czibula, y Bocicor (2011). Jafari y cols.

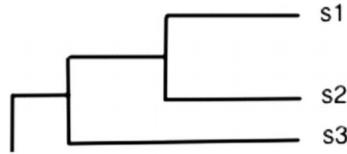


Figura 2.16: Este tipo particular de árbol guía se puede visualizar mediante una permutación de las secuencias.

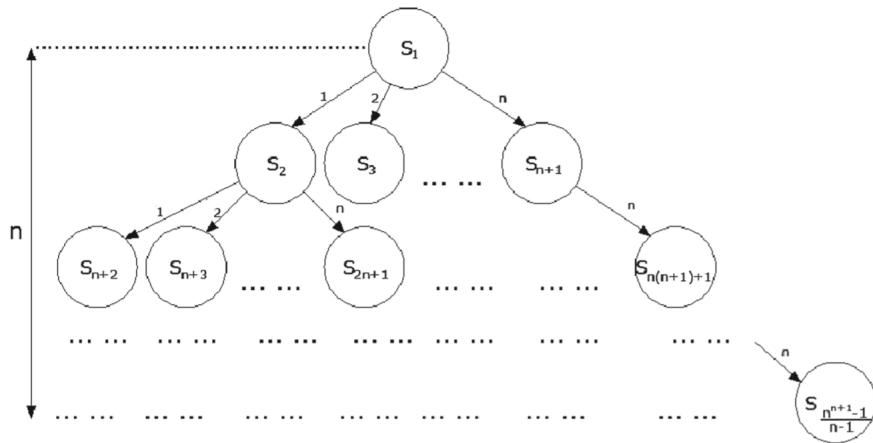


Figura 2.17: Estados y acciones del enfoque de 2.16.

(2019) utilizan el mismo esquema estado-acción-recompensa de Mircea y cols. (2014), pero utiliza deep Q-Learning with experience replay sobre el enfoque. Los autores de los trabajos anteriores consiguieron resultados de calidad equiparable a los métodos CLUSTALW y MAFFT en un menor tiempo computacional. Por tanto, sugieren el estudio de nuevos métodos basados en aprendizaje reforzado para abordar este problema.

Capítulo 3

Enfoques de RL propuestos

En este Capítulo se proponen 3 enfoques para abordar el problema del MSA usando RL. Dos de ellos se basan en la estrategia general del alineamiento progresivo extendiendo el enfoque de [Mircea y cols. \(2014\)](#), mientras que el tercer enfoque es una extensión del algoritmo de Needlman-Wunsch para más de dos secuencias. Designaremos a estos métodos como el enfoque de permutaciones, árboles y sufijos, respectivamente.

Las principales características de los tres enfoques a analizar son las siguientes:

1. **Enfoque de permutaciones:** Se basa en la estrategia del alineamiento progresivo. Al igual que los enfoques de [Mircea y cols. \(2014\)](#) y [Jafari y cols. \(2019\)](#) el espacio de búsqueda está conformado por aquellos alineamientos que se obtienen mediante permutar el orden de las secuencias y hacer el alineamiento progresivo siempre eligiendo una secuencia nueva que alinear con el resultado de las secuencias previamente elegidas. La propuesta en este enfoque consiste en reducir el conjunto de estados válidos a solo aquellos que representan un proceso de alineamiento progresivo válido.
2. **Enfoque de árboles:** También se basa en la estrategia del alineamiento progresivo, pero a diferencia del enfoque anterior, el espacio de búsqueda consiste de todos los árboles filogenéticos que pudieron dar origen a las secuencias.

3. **Enfoque de sufijos:** Este enfoque se realiza sobre un espacio de búsqueda mucho más intensivo que los anteriores, es una extensión del algoritmo Needleman-Wunsch para el caso de múltiples secuencias se construyen alineamientos óptimos entre sufijos de las secuencias de entrada.

3.1. Alineamiento Progresivo sobre el espacio de permutaciones de secuencias

La primera extensión propuesta consiste en modificar el espacio de estados del método descrito en [Mircea y cols. \(2014\)](#): el espacio de estados se reduce para sólo considerar estados que representen el prefijo de alguna permutación. Asimismo, las acciones desde cada estado se restringen a aquellas que llevan a un estado válido. De forma general, el paradigma estado-acción-recompensa se puede resumir:

- **Estados:** Son k -adas ordenadas $0 \leq k \leq N$ cuyos elementos son números enteros distintos entre 1 y N . De esta forma, cada estado corresponde a un prefijo de una permutación de los índices de las secuencias y representa pasos intermedios en la elección de algún alineamiento progresivo asociados a una permutación en particular. El estado inicial es una 0-ada o conjunto vacío. Todas las permutaciones de los números entre 1 y N son estados finales. La cantidad total de estados se puede calcular dividiendo sobre la longitud de las k -adas, una vez fija una k , la cantidad de formas de elegir las secuencias que conforman la k -ada son $\binom{N}{k}$, cada una de estas elecciones se puede ordenar de $k!$ formas distintas, entonces la cantidad de estados es

$$\begin{aligned} \sum_{k=0}^N \binom{N}{k} k! &= \sum_{k=0}^N \frac{N!}{(N-k)!} = N! \sum_{k=0}^N \frac{1}{(N-k)!} \\ &= N! \sum_{k=0}^N \frac{1}{k!} \approx N! \times e, \end{aligned}$$

donde la última igualdad se da al reacomodar los términos de la suma y la

última aproximación se sigue de la identidad $\sum_{n=0}^{\infty} \frac{1}{n!} = e$.

- **Acciones:** Desde un estado no final, las acciones asociadas consisten en la elección de la siguiente secuencia a incluir en el alineamiento. Es decir, es la elección de un índice $1 \leq i \leq N$ para concatenar al estado actual s , restringido a que i no estuviera ya presente en s . Luego, desde un estado válido de tamaño k se tienen $N - k$ posibles acciones.
- **Recompensa:** Sea s un estado no final y a una acción válida desde s . Si s es el conjunto vacío, la recompensa es 0; en caso contrario, asociado a s se tiene un alineamiento de algunas (posiblemente solo una) secuencias, entonces se procede a realizar el alineamiento mediante Needleman-Wunsch y se asigna como recompensa el SP-score del nuevo alineamiento generado restringido a la secuencia nueva que fue añadida según la acción a .

El enfoque anterior se ejemplifica en la Figura 3.1. Para la implementación de este enfoque, computacionalmente cada estado es un vector de tamaño fijo N , cuyas entradas son enteros entre 0 y N . Los 0 reflejan la ausencia de elección, esto queda ejemplificado en la Figura 3.2

3.2. Alineamiento Progresivo sobre el espacio de árboles filogenéticos

Considerando que algoritmos como CLUSTALW o MAFFT realizan alineamientos progresivos sobre una variedad de árboles más extensa (ver ejemplo en la Figura 3.3), se considera el siguiente enfoque:

- **Estados:** Cada estado representa los agrupamientos intermedios que se van presentando al seguir un árbol. Para mayor formalidad de este conjunto de estados, describiremos algunas características que presentan los árboles válidos. Un árbol enraizado válido posee las siguientes características:
 - Tiene N hojas.

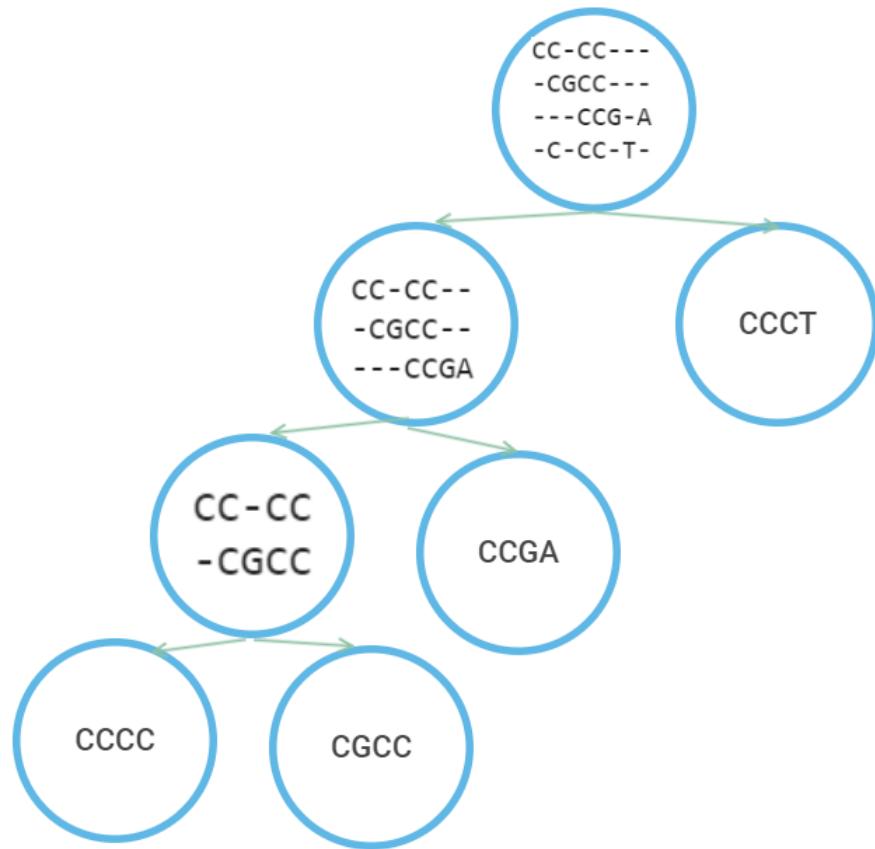


Figura 3.1: [Enfoque de permutaciones]. Ejemplo de alineamiento progresivo sobre 4 secuencias usando NW para los alineamientos intermedios. Tomando un scoring de +4 para match, -4 para mismatch entre letras y -1 para letra con gap.

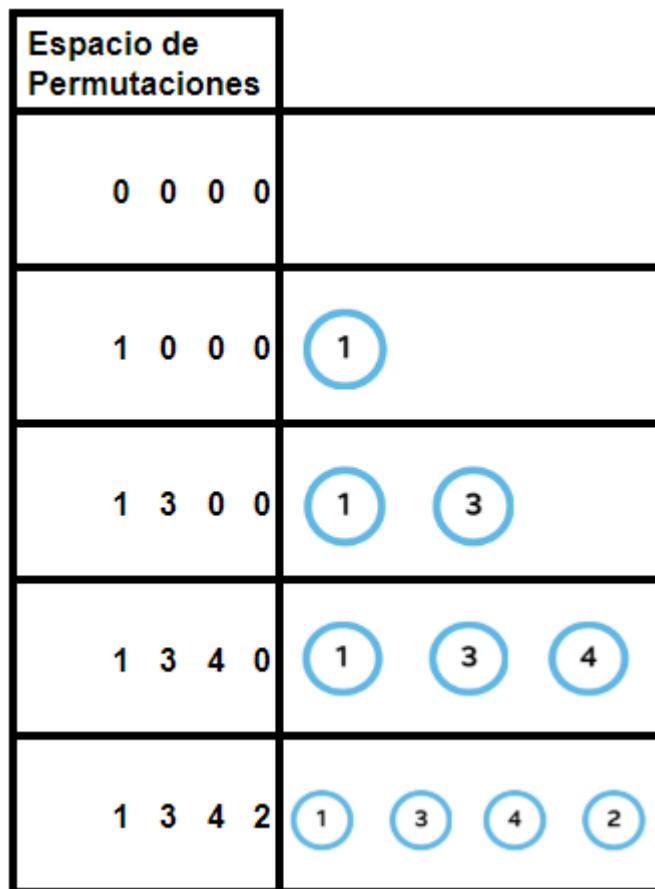


Figura 3.2: [Enfoque de permutaciones]. Representación computacional de un episodio, la sucesión de acciones a las que corresponde es: Elegir secuencia 1, elegir secuencia 3, elegir secuencia 4 elegir secuencia 4.

- Todos los nodos que no son hojas ni raíz tienen 2 hijos y un parente.
- La raíz tiene 2 hijos.

Cormen, Leiserson, Rivest, y Stein (2009) mencionan las siguientes dos propiedades de los árboles enraizados:

- La suma de la cantidad de aristas que salen de cada nodo es igual a dos veces la cantidad de aristas. Es decir, en un grafo G no dirigido con nodos v_1, v_2, \dots, v_V y M aristas, se tiene que si $\deg(v)$ denota la cantidad de aristas que salen del nodo v , entonces

$$\sum_{i=1}^V \deg(v_i) = 2M.$$

- Un arbol con V nodos, tiene $V - 1$ aristas.

De la información anterior se tiene que si un árbol válido (con N hojas, donde cada hoja representa una de las N secuencias) tiene x nodos que no son hoja ni raíz, entonces la suma de aristas que salen de los nodos es $N + 3x + 2$ y la cantidad de nodos es $N + x + 1$, por tanto se tiene que:

$$N + 3x + 2 = 2(N + x + 1 - 1)$$

de donde $x = N - 2$. De tal forma que en un árbol válido enraizado se tienen $x + 1 = N - 1$ nodos con 2 hijos. Un aspecto importante a considerar es que en la práctica no importa quién es el hijo derecho y quien el hijo izquierdo, por tanto un mismo árbol puede representar alineamientos equivalentes, específicamente cada nodo con 2 hijos proporciona dos opciones para considerar, por tanto se tiene una representación de 2^{N-1} alineamientos. Luego, si consideramos como f_n a la cantidad de árboles válidos con n hojas no etiquetadas, la cantidad de alineamientos asociados son

$$\frac{N! \times f_N}{2^N},$$

donde multiplicamos por $N!$ pues es la cantidad de formas en las que podemos asignar las N secuencias a las N hojas. Podemos realizar el conteo de f_n de forma recursiva de la siguiente manera

$$f_n = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} f_k \times f_{n-k} & n > 1 \end{cases}$$

pues con una sola hoja se tiene solamente un árbol válido, mientras que con $n > 2$ hojas, podemos hacer el conteo dividiendo en casos disjuntos sobre la cantidad de hojas k que estarán en la rama izquierda del árbol; de esta forma $n - k$ hojas estarán en la rama derecha y la construcción de estas ramas se pueden realizar respectivamente de f_k y f_{n-k} maneras distintas. Como se explica en Cormen y cols. (2009), la recursión que sigue f_n es la recursión de los números de Catalán, y por tanto se infiere la siguiente identidad

$$f_N = \frac{\binom{2(N-1)}{N-1}}{N},$$

de donde podemos entonces concluir que la cantidad de alineamientos representados por los árboles son

$$\begin{aligned} \frac{N! \binom{2(N-1)}{N-1}}{2^{N-1}} &= \frac{N!(2N-2)!}{(N-1)!(N-1)!N2^{N-1}} = \frac{(2N-2)!}{(N-1)!2^{N-1}} \\ &= \frac{(2N-2)!}{(2N-2)(2N-4) \times \cdots \times 4 \times 2} = (2N-3) \times (2N-5) \times \cdots \times 3 \times 1. \end{aligned}$$

Para la representación computacional de los estados, se consideran vectores de longitud N , donde la entrada $1 \leq i \leq N$ es el representante del cluster actual de la secuencia i , dicho representante siempre es el menor índice de las secuencias en el cluster. Existe un único estado inicial que corresponde al vector $(1, 2, \dots, N)$, ya que de inicio cada secuencia pertenece a un cluster individual. El único estado final corresponde al estado $(1, 1, \dots, 1)$ pues ese estado representa que todas las secuencias ya forman un solo cluster. Notemos que en la i -ésima coordenada

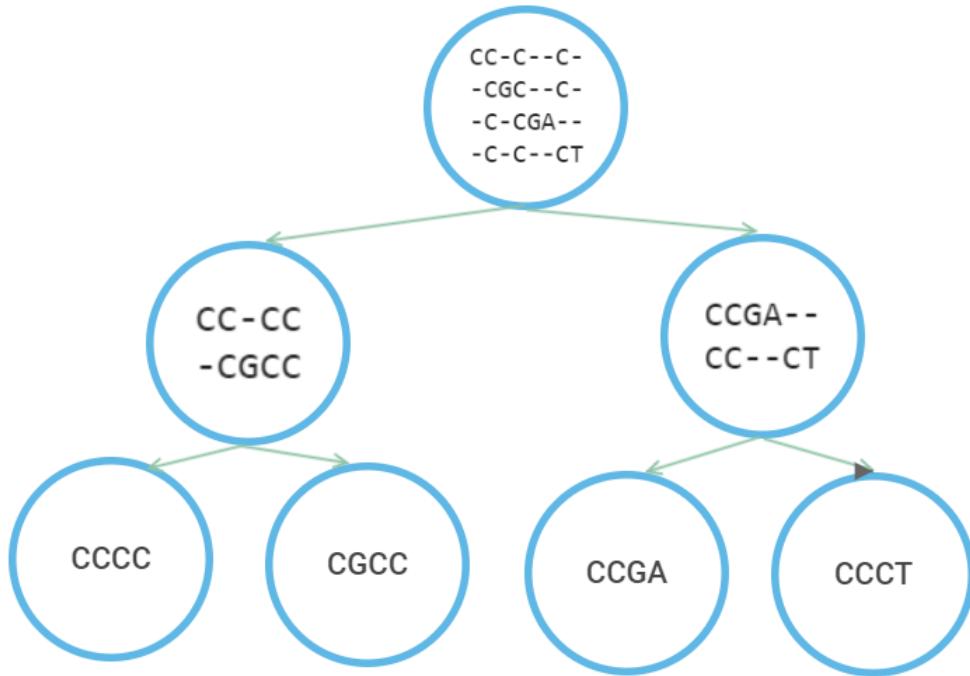


Figura 3.3: [Enfoque de árboles]. Ejemplo de alineamiento progresivo sobre 4 secuencias usando NW para los alineamientos intermedios. En este caso esta configuración no corresponde a ninguna permutación de las secuencias. Se tomo el mismo scoring de la Figura 3.1.

en ningún momento puede tomar un valor mayor que i , pues por definición el representante del cluster tiene que ser el menor índice de entre los índices que conforman el cluster. Luego, la cantidad de estados esta acotada por $N!$.

- **Acciones:** Desde un estado no final s , las acciones válidas consisten en elegir 2 clusters disjuntos para fusionarse. Así, la cantidad de acciones esta acotada por $\binom{N}{2}$.
- **Recompensa:** Desde un estado no final s y una acción a la recompensa observada consiste en el SP-score del alineamiento que se obtiene de alinear los perfiles asociados a los clusters a unir restringido a parejas de filas en clusters distintos.

El método del enfoque de árboles se ejemplifica en la Figura 3.4 y la Figura 3.5.

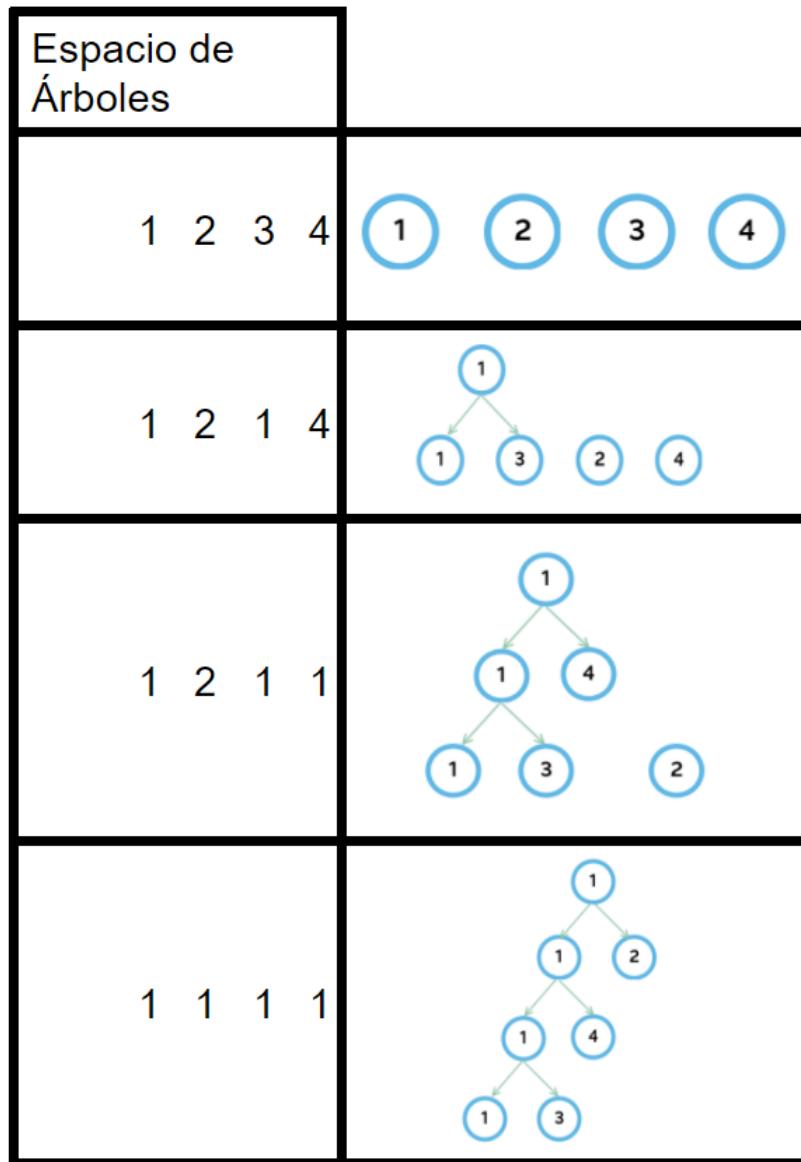


Figura 3.4: [Enfoque de árboles]. Inicialmente se tienen 4 clusters, posteriormente se toma la acción de unir los clusters 1 y 3. De esta forma el representante de la secuencia 3 es el índice 1. Posteriormente se une el cluster representado por 1 con el cluster representado por 4 y finalmente se añade la secuencia 2.

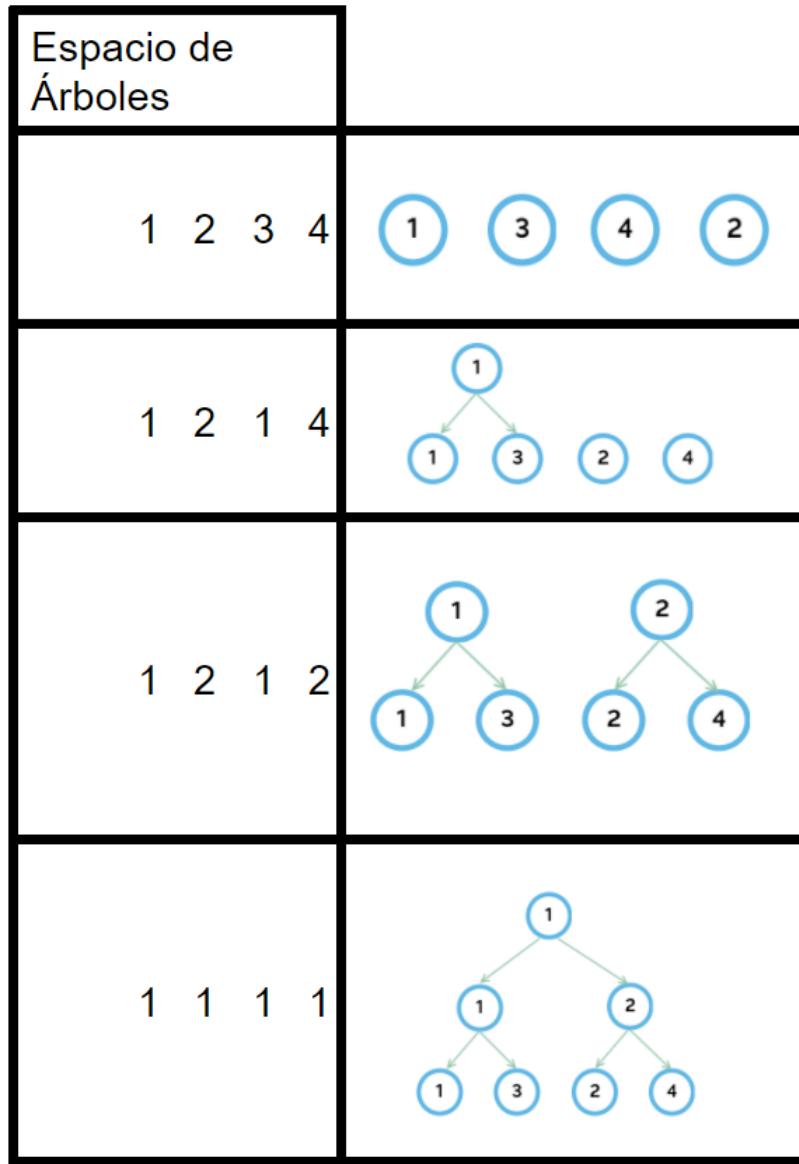


Figura 3.5: [Enfoque de árboles]. Inicialmente se tienen 4 clusters, posteriormente se toma la acción de unir los clusters 1 y 3. De esta forma el representante de la secuencia 3 es el índice 1. Posteriormente se une el cluster representado por 2 con el cluster representado por 4 y finalmente se unen los dos clusters presentes.

3.3. Extensión de NW sobre el espacio de sufijos de secuencias

A diferencia de los casos anteriores, este enfoque no está basado en el alineamiento progresivo, con él se busca construir un alineamiento considerando todas las secuencias simultáneamente, en forma análoga al algoritmo de Needleman-Wunsch para alinear parejas de secuencias. Así, el paradigma estado-acción-recompensa es el siguiente:

- **Estados:** Cada estado está representado por N sufijos de las N secuencias. En el estado inicial, dichos sufijos corresponden a las secuencias completas, los estados intermedios pueden incluir sufijos vacíos, esto representa que la secuencia en cuestión ya fue completamente alineada previamente. El único estado final es el formado por N sufijos vacíos. Computacionalmente, los estados se representan como vectores N -dimensionales cuyas entradas para la i -ésima coordenada son los enteros entre 0 y la longitud l_i de la i -ésima secuencia; esto es, las entradas del vector representan cuántos de los caracteres de cada una de las N secuencias han sido colocados en el alineamiento a construir. De esta forma, la cantidad total de estados es $(l_1 + 1) \times (l_2 + 1) \times \cdots \times (l_N + 1)$; tomando $L = \max_i(l_i)$ una cota de la cantidad de estados es L^N .
- **Acciones:** Dado un estado no final s , las acciones corresponden a decidir sobre qué secuencias (de entre las que aún no se han procesado completamente) incluir un nuevo carácter, esto lleva a un nuevo estado que corresponde a los sufijos que conforman s de las secuencias sobre las cuales se decidió colocar un gap y a los sufijos removiendo el primer carácter de todas las secuencias sobre las cuales se toma la decisión de no incluir un gap. Dentro de las opciones, no se contempla el caso de no avanzar en ninguna secuencia, por tanto la cantidad total de acciones es $2^N - 1$, pues 2^N corresponde al conteo de todas las posibles maneras de elegir sobre cada secuencia avanzar o no, y se debe restar el estado de no avanzar sobre ninguna secuencia.
- **Recompensa:** Tomar una acción a sobre un estado s , genera una nueva co-

lumna en el alineamiento que se está obteniendo, de esta forma la recompensa observada se toma como el SP-score de la nueva columna.

En la Figura 3.6 podemos ver los primeros pasos de un episodio bajo este enfoque. Como se menciona en el Capítulo 4 y en el trabajo futuro, aún se trabaja para lograr resultados aceptables empleando este enfoque.

3.4. Comparaciones entre los enfoques

Como se mencionó anteriormente, el método de árboles filogénéticos es una generalización del método de permutaciones. Así, surgen algunas preguntas naturales en la comparación de ambos algoritmos:

- **¿Existen casos donde el máximo score de alineamiento progresivo sobre permutaciones sea menor que el máximo score que se pueda obtener sobre el espacio de árboles filogenéticos?** La respuesta es sí. Bajo un esquema de score de +4 para match, -4 para mismatch entre letras y -1 para letra con gap sobre las siguientes 5 secuencias:

```
CCCCGAACCTCCTCCCTGGTGCTCTCAGGGGTCTGGCCCTG
CGCCCCGAACCTCCTCCCTGGTGCTCTCAGGGGCCCTGGCCCTGACCCAGACCCGGGCAGGCTCTCACTCCATGAGCTATT
CCGAACCTCCTCCCTGGTGCTCTCAGGGGCCCTGGCCCTGACCCAGACCTGGCGAGCTCCACTCCA
CCCTGACCGAGAC
CCTGACCGAGACCTGGGCCGGCTCGCACTCCATGAAGTATTCTACAC
```

Se tiene, por inspección, que el máximo score que se puede obtener sobre el espacio de permutaciones es 720, mientras que sobre el espacio de árboles filogenéticos es 732.

De esta forma, se justifica la expansión del espacio de búsqueda de la propuesta original de Mircea et al (2014) para tratar de encontrar un alineamiento con un score mejor.

- **¿Que tanto se expande el espacio de búsqueda?** Como se mencionó antes,

Estado	0	1	2	3	4	5	6	Alineamiento	A.P	A.F		
0		A	G	T	T	G			NA	1		
0			G	T	C	C			NA	0		
0				T	T	A	C		NA	1		
0					G	G	C	T	G	T	NA	0
0						T	T	A	C		NA	0

Estado	0	1	2	3	4	5	6	Alineamiento	A.P	A.F			
1		A	G	T	T	G		A	1	1			
0			G	T	C	C		-	0	1			
1				T	T	A	C	T	1	0			
0					G	G	C	T	G	T	-	0	0
0						T	T	A	C	-	0	1	

Estado	0	1	2	3	4	5	6	Alineamiento	A.P	A.F			
2		A	G	T	T	G		A G	1	0			
1			G	T	C	C		- G	1	1			
1				T	T	A	C	T -	0	1			
0					G	G	C	T	G	T	- -	0	1
1						T	T	A	- T		-	1	0

Estado	0	1	2	3	4	5	6	Alineamiento	A.P	A.F	
2		A	G	T	T	G		A G -	0	1	
2			G	T	C	C		- G T	1	1	
2				T	T	A	C	T - T	1	1	
1					G	G	C	T	- - G	1	1
1						T	T	A	- T -	0	1

Figura 3.6: [Enfoque de sufijos]. Considerando el alineamiento de las secuencias $AGTTG$, $GTCC$, $TTAC$, $GGCTGT$, y $TTAC$, se tiene que el estado inicial es el vector $(0, 0, 0, 0, 0)$, esto representa que ningún carácter ha sido procesado. A.P denota la acción previa, A.F denota la acción futura. Al tomar la acción $(1, 0, 1, 0, 0)$, se toman los próximos caractéres por procesar de la primer y tercera secuencia, formando la columna $A - T --$, posteriormente a esto se opta por tomar la acción $(1, 1, 0, 0, 1)$ para anexar la columna $GG -- T$ y posteriormente se toma la acción $(0, 1, 1, 1, 0)$ para agregar la columna $- TTG -$, el resto del episodio se sigue de esta manera hasta llegar al estado final $(5, 4, 4, 6, 4)$.

n	Permutaciones	Árboles	Árboles/Permutaciones
3	3	3	1
4	12	15	1.25
5	60	105	1.75
6	360	945	2.625
7	2520	10395	4.125
8	20160	135135	6.7031
9	181440	2027025	11.1718
10	1814400	34459425	18.9921

Tabla 3.1: Cantidad de opciones en cada enfoque.

la cantidad de procesos que generan alineamientos sobre el espacio de permutaciones es $\frac{N!}{2}$ y sobre el espacio de árboles filogenéticos es $(2N - 3) \times (2N - 5) \times \cdots \times 3 \times 1$. En la Tabla 3.1 podemos observar la cantidad de procesos para los primeros 7 casos de interés ($3 \leq N \leq 10$), donde podemos notar que conforme N crece, la diferencia se vuelve cada vez más notoria, de hecho si $P(N) = \frac{N!}{2}$ y $T(N) = (2N - 3) \times (2N - 5) \times \cdots \times 3 \times 1$ denotan la cantidad de procesos distintos para el enfoque de permutaciones y de árboles respectivamente, entonces se tiene que

$$\begin{aligned} \frac{T(N)}{P(N)} &= \frac{(2N - 3) \times (2N - 5) \times \cdots \times 3 \times 1}{\frac{N!}{2}} \\ &\geq \frac{(2N - 4) \times (2N - 6) \times \cdots \times 2 \times 1}{\frac{N!}{2}} = \frac{2^{N-2}(N-2)!}{\frac{N!}{2}} \\ &= \frac{2^{N-1}(N-2)!}{N \cdot (N-1) \cdot (N-2)!} = \frac{2^{N-1}}{N(N-1)}. \end{aligned}$$

Luego, $\frac{T(N)}{P(N)}$ presenta un crecimiento exponencial, como podemos constatar en la tercera columna de la Tabla 3.1.

- **¿En un tiempo acotado, la solución del método de árboles es al menos tan buena como la del método de permutaciones?** Esta pregunta se abordará a mayor detalle en el Capítulo 4, Sección 4.4.

Sobre el espacio de sufijos, notemos que la duración máxima de un episodio se

	Número de Estados	Número de Acciones	Duración Episodio
Permutaciones	$N! \times e$	N	N
Árboles	$N!$	$\binom{N}{2}$	$N - 1$
Sufijos	L^N	$2^N - 1$	$N \times L$

Tabla 3.2: Asumiendo que se alinean N secuencias de longitud menor o igual que L , la tabla presenta cotas para la cantidad de estados y acciones y la duración de un episodio para cada uno de los enfoques descritos en este capítulo.

obtiene al avanzar un carácter a la vez, de esta forma la duración máxima es

$$l_1 + l_2 + \dots + l_N \leq N \times L.$$

Teniendo esto en cuenta y lo desarrollado anteriormente, resumimos cotas sobre el número de estados y acciones y la duración de los episodios para los tres enfoques propuestos en esta tesis.

En la Tabla 3.2 se puede observar que los episodios bajo el enfoque de alineamiento progresivo son muy cortos. Por ejemplo, para alinear 5 secuencias, los episodios tendrán 5 y 4 transiciones, respectivamente. Sin embargo, si las 5 secuencias tienen alrededor de 1000 caracteres, entonces un episodio del método de sufijos puede tener hasta 5000 transiciones. Lo anterior puede considerarse un indicador de que el enfoque de sufijos puede ser más eficiente sobre cadenas de pequeña/mediana longitud; este aspecto también se abordará con mayor detalle en el Capítulo 4.

Capítulo 4

Análisis del rendimiento de los enfoques propuestos

En este capítulo se realizan pruebas a los métodos propuestos con la finalidad de analizar su eficacia y eficiencia. Para esto, se consideran los 7 conjuntos de datos utilizados por [Mircea y cols. \(2016\)](#). Dichos conjuntos fueron también utilizados por [Jafari y cols. \(2019\)](#) y en ambos trabajos se realizó una comparativa de sus respectivos métodos contra dos de los métodos más usados CLUSTALW y MAFFT.

La fuente para dos de los conjuntos mencionados anteriormente es [Carroll y cols. \(2007\)](#). Este conjunto contiene un repositorio de alineamientos múltiples de secuencias de ADN, cuyos alineamientos fueron obtenidos a partir de la estructura de las proteínas que codifican. Este conjunto de datos nos será de utilidad pues contiene cientos de conjuntos de secuencias con un alineamiento propuesto. Se realizó una comparación de los alineamientos de dicha base de datos con los generados por los métodos propuestos en este trabajo analizando algunas métricas de los mismos y utilizando el score GUIDANCE de [Penn, Privman, Landan, Graur, y Pupko \(2010\)](#). El score GUIDANCE presenta una medición de la robustez de los alineamientos obtenidos con respecto a la incertidumbre del árbol guía utilizado.

Las pruebas anteriores fueron utilizadas en los métodos basados en alineamiento

progresivo (sobre permutaciones de secuencias y sobre árboles filogenéticos) utilizando *Deep Q-learning with experience replay* con **resultados positivos** en cuanto a la calidad de los alineamientos obtenidos. Los resultados no fueron satisfactorios con el método que trabaja el espacio de sufijos. Debido a esto, dicho enfoque se abordó utilizando el algoritmo *Neural Fitted Q Iteration* pues dicho algoritmo resulta de utilidad cuando es posible obtener múltiples experiencias (posiblemente simuladas) sin la necesidad de simular episodios completos; si bien se observó una mejora en el desempeño del método, los resultados continuaron siendo no satisfactorios.

4.1. Datos

Los 7 conjuntos de datos mencionados al inicio del capítulo provienen de tres fuentes:

1. El Laboratorio Europeo de Biología Molecular (EMBL, por sus siglas en inglés). En [Kanz y cols. \(2005\)](#) podemos observar como obtener secuencias de la base de datos de la EMBL utilizando su número de acceso del Archivo Europeo de Nucleótidos (ENA, por sus siglas en inglés).
2. El conjunto de alineamientos múltiples de secuencias de nucleótidos generado por [Carroll y cols. \(2007\)](#). En particular se utilizan conjuntos de secuencias del archivo `oxbench_mds_all`.
3. El artículo de [Xiang, Zhang, Qin, y Fu \(2010\)](#), donde se propone un algoritmo genético para alineamiento múltiple de secuencias de ADN utilizando representaciones de los alineamientos en un grafo plano y el algoritmo Ant Colony.

Cada uno de los conjuntos de secuencias alineados para comparar el desempeño de los algoritmos propuestos aquí, tienen características particulares que los hacen heterogéneos. Por ejemplo, cada conjunto de datos tiene una longitud distinta al resto, algunas de estas secuencias codifican un gen particular, otras provienen de organismos distintos, entre otras propiedades. A continuación presentamos el nombre de cada

uno de estos conjuntos de datos junto con una breve descripción de algunas de las características que conocemos sobre ellos ¹.

HC: Virus del Hepatitis C (subtipo 1b) obtenido de la EMBL. Son 10 secuencias de longitud promedio 211.9, los números de acceso de la ENA son Z75841, Z75850, Z75851, Z75852, Z75853, Z75854, Z75858, Z75859, Z75860, Z75861.

PA: 5 secuencias del organismo *Papio anubis* obtenido de la EMBL. La longitud promedio de las secuencias es de 1093 nucleótidos. Los números de acceso de la ENA son U35624, U35625, U35626, U35627, U35628.

OX469: Es el conjunto 469 del archivo oxbench_ mdsa_all. Se conforma por 3 secuencias de longitud promedio 332.

OX429: Es el conjunto 429 del archivo oxbench_ mdsa_all. Se conforma por 12 secuencias de longitud promedio 171.

D1Ant: Es el primer conjunto de datos utilizado en [Xiang y cols. \(2010\)](#). Se conforma por 3 secuencias de longitud promedio 39.33.

LGM: Segundo conjunto de [Xiang y cols. \(2010\)](#), se compone de 3 secuencias de ADN provenientes del lemur, gorila y ratón. La longitud promedio de las tres secuencias es de 93 nucleótidos.

RLO: Tercer conjunto de [Xiang y cols. \(2010\)](#), se compone de 3 secuencias de ADN provenientes del gen de la *beta globina* de la rata, el lémur y la zarigüeya. La longitud promedio de las secuencias es de 129 nucleótidos.

En la Tabla 4.1 podemos encontrar un resumen de las dimensiones de los conjuntos anteriores.

¹Es importante mencionar que no contamos con todos los detalles biológicos de estos conjuntos de datos. Las fuentes de ellos contienen información para que alguien experto en la nomenclatura de los formatos usados para guardar esta información pueda inferir todos los detalles genéticos y biológicos involucrados. En este trabajo no se consideró necesario completar todos estos detalles pues no son relevantes para el análisis de rendimiento.

Data Set	N	L promedio	L máx	L mín
HC	10	211.90	212	211
PA	5	1093.00	1098	1088
469OX	3	332.00	339	321
429OX	12	171.00	183	153
D1Ant	3	39.33	40	39
LGM	3	93.00	94	92
RLO	3	129.00	129	129

Tabla 4.1: Dimensiones de los distintos conjuntos de datos utilizados: N representa la cantidad de secuencias y L hace referencia a la cantidad de nucleótidos de las secuencias.

4.2. Deep Q-Learning con enfoques basados en alineamiento progresivo

Para ajustar la función Q y obtener una política con la cual generar un alineamiento, se utilizó el Algoritmo 2, Deep Q-Learning with experience replay, el cuál fue explicado en el Capítulo 2 sobre los enfoques de alineamiento progresivo. Tanto para el caso de espacio de permutaciones como para el caso de espacio de árboles, se utilizó como aproximador de Q una red neuronal multientrada; en la Figura 4.1 se puede ver una esquematización de las distintas capas utilizadas. Los detalles de la implementación se encuentran en el Anexo B.

En la experimentación se dejaron fijos algunos de los hiperparámetros del algoritmo, la elección de dichos parámetros se realizó en forma heurística. Además, para alinear un conjunto de N secuencias, se encontraron los mejores resultados con las siguientes consideraciones:

- Se usa el mismo scoring empleado por [Mircea y cols. \(2016\)](#) y [Jafari y cols. \(2019\)](#), tomando un score de +2 cuando dos nucleótidos iguales son alineados, -1 cuando nucleótidos distintos son alineados y -2 cuando un nucleótido es alineado con un gap.
- El método se detiene después de pasar $2 \times N$ episodios sin mostrar una mejora en el SP-score correspondiente al alineamiento que se obtiene de seguir la política

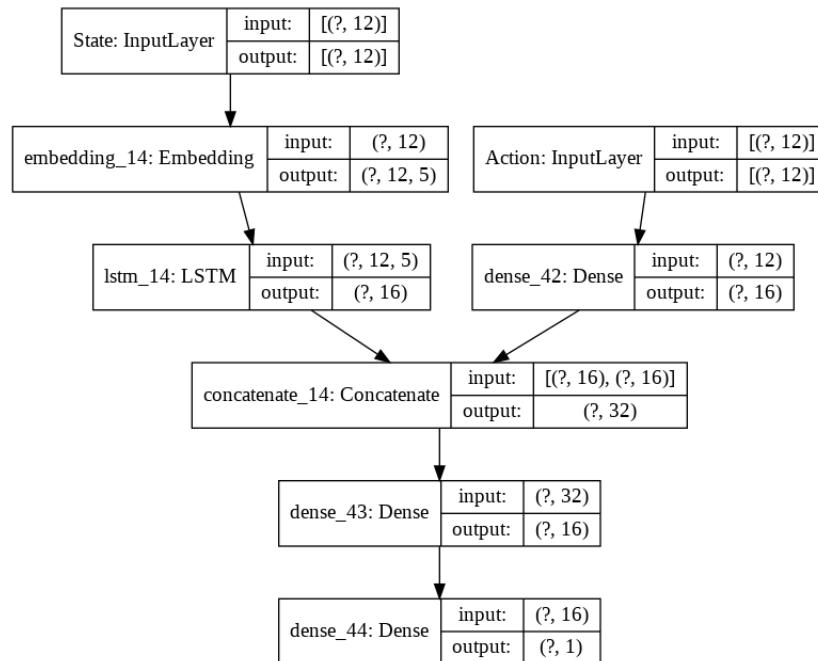


Figura 4.1: Ejemplo de la red utilizada para el conjunto OX429 con 12 secuencias. La red toma como entrada un batch de estados con sus respectivas acciones representadas como vectores con 12 entradas. Posteriormente, el estado es procesado por una capa de embedding y una red LSTM (ver Anexo B) para producir un vector de dimensión 16. La acción es pasada por una capa densa para producir un vector de dimensión 16. Posteriormente, los vectores anteriores se concatenan para producir un vector de dimensión 32 que representa en conjunto el estado-acción sobre el cual se quiere calcular su Q valor. Para esto se utiliza una capa densa intermedia para producir una salida de dimensión 16 y finalmente producir la aproximación del Q valor deseado.

greedy del estado actual del aproximador de Q .

- Se regresa el alineamiento y el aproximador de Q cuyo SP-score fue máximo.
- La cantidad máxima de episodios a explorar es N^2 .
- El repositorio de experiencias se conforma de las últimas N^2 experiencias obtenidas.
- Cada que se actualiza la red, se toma un batch aleatorio de $2 \times N$ experiencias durante $2 \times N$ épocas.
- Se toma $\alpha = 1$ y $\epsilon = 0.3$ como parámetros en el Algoritmo 2.

Para el hiperparámetro γ se ejecutó el algoritmo utilizando $\gamma = 0.01, 0.1, 0.5, 0.9, 0.99$; y se reportó el modelo con mejor rendimiento.

Además del SP-score, también se revisan las siguientes métricas:

AL: Longitud del alineamiento.

EM: Cantidad de columnas que contienen un solo nucleótido.

CS: Column score, $CS = EM/AL$

Observaciones

De los resultados mostrados en la Tabla 4.2, podemos notar que cuando se tienen pocas secuencias el valor de γ no resulta determinante. Esto hace sentido pues la longitud de los episodios es muy corta. Al contrario, para el conjunto de 12 secuencias, donde un gamma grande obtuvo mejores resultados. De igual manera es relevante notar que con pocas secuencias no se aprecia una diferencia en tiempos de ejecución entre los enfoques de permutaciones y árboles, para casos con más secuencias el tiempo de ejecución del enfoque de los árboles tiende a ser mayor, esto debido a que muestra mejorías durante mayor cantidad de episodios con respecto al enfoque de

permutaciones. En la Figura 4.2-4.15 se muestra el histórico de la evolución del SP-score y el column score para los 7 conjuntos de datos y el enfoque de permutaciones y de árboles.

4.2.1. Comparación con MAFFT y CLUSTALW

En esta sección se comparan los algoritmos propuestos contra los métodos MAFFT y CLUSTALW. Para realizar esta comparación, utilizamos los datos empleados en [Mircea y cols. \(2016\)](#) y [Jafari y cols. \(2019\)](#) para mostrar el desempeño de los métodos ahí propuestos. Estos resultados se resumen en la Tabla 4.3, donde podemos notar que, con excepción del conjunto HC (donde todos los métodos llegan al mismo alineamiento) el SP-score máximo se obtiene con alguno de los métodos propuestos. En particular el enfoque de árboles obtiene siempre el máximo column score.

4.2.2. Comparación contra enfoques de Mircea y Jafari

Ahora contrastaremos los enfoques propuestos en este trabajo contra los resultados obtenidos por las propuestas basadas en aprendizaje reforzado de [Mircea y cols. \(2016\)](#) y [Jafari y cols. \(2019\)](#). Dichos resultados se resumen en la Tabla 4.4, donde podemos observar que el enfoque de permutaciones mantiene o mejora el SP-score en todos los casos con respecto a las otras propuestas de Mircea y Jafari. El enfoque de árboles a su vez iguala o supera los resultados obtenidos por el enfoque de permutaciones con excepcion en el conjunto RLO. También podemos notar que, si bien el column score en las propuestas Mircea y Jafari son mayores en los conjuntos 469OX, 429OX y LGM, las diferencias en el SP-score indican que las columnas no alineadas perfectamente son alineadas en forma más eficiente sin sacrificar una cantidad excesiva de columnas perfectamente alineadas.

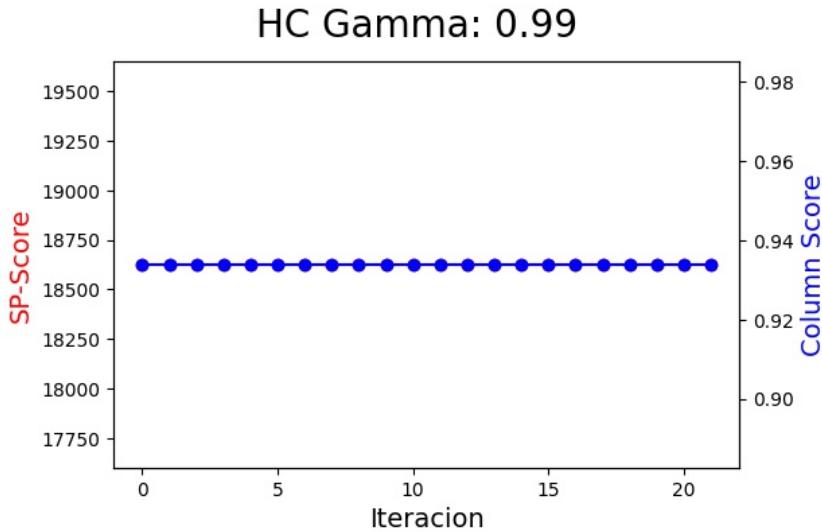


Figura 4.2: [Permutaciones, HC, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de permutaciones. Desde el primer episodio se obtiene estabilidad. No se ve el color rojo pues esta justo por debajo del color azul.

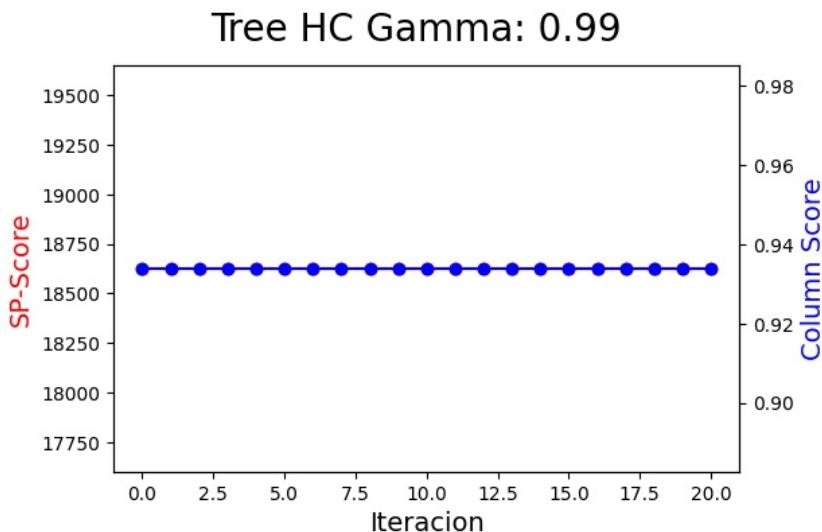


Figura 4.3: [Arboles, HC, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de árboles. Desde el primer episodio se obtiene estabilidad. No se ve el color rojo pues esta justo por debajo del color azul.

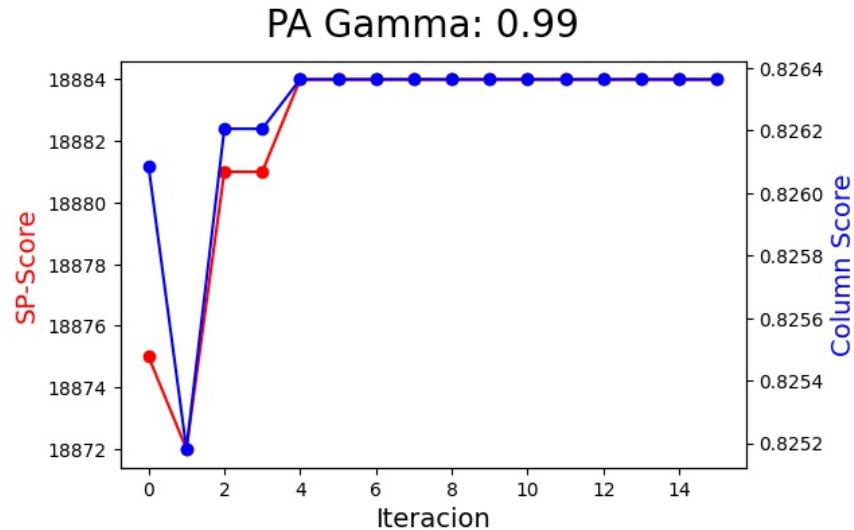


Figura 4.4: [Permutaciones, PA, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de permutaciones. Se alcanza estabilidad desde el episodio 5.

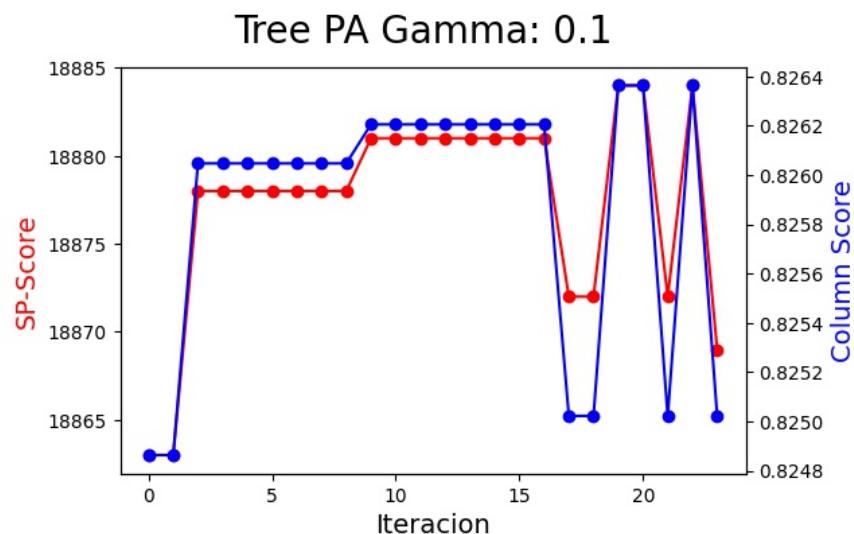


Figura 4.5: [Árboles, PA, $\gamma = 0.1$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de árboles. Se alcanza el máximo por primera vez en el episodio 20.

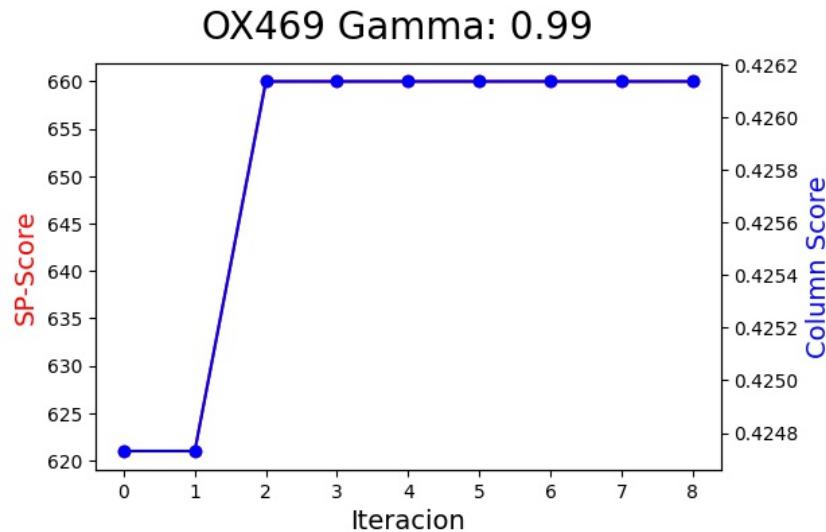


Figura 4.6: [Permutaciones, OX469, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de permutaciones. Se alcanza estabilidad desde el tercer episodio.

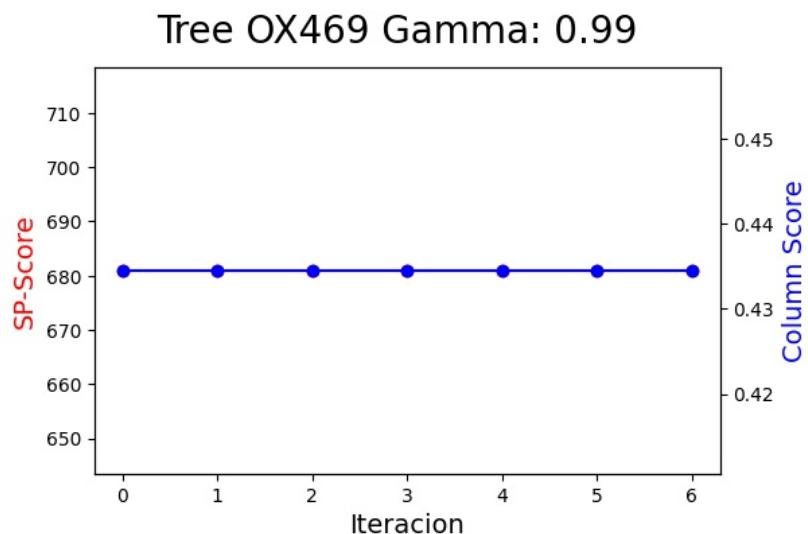


Figura 4.7: [Árboles, OX469, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de árboles. Se alcanza estabilidad desde el primer episodio.

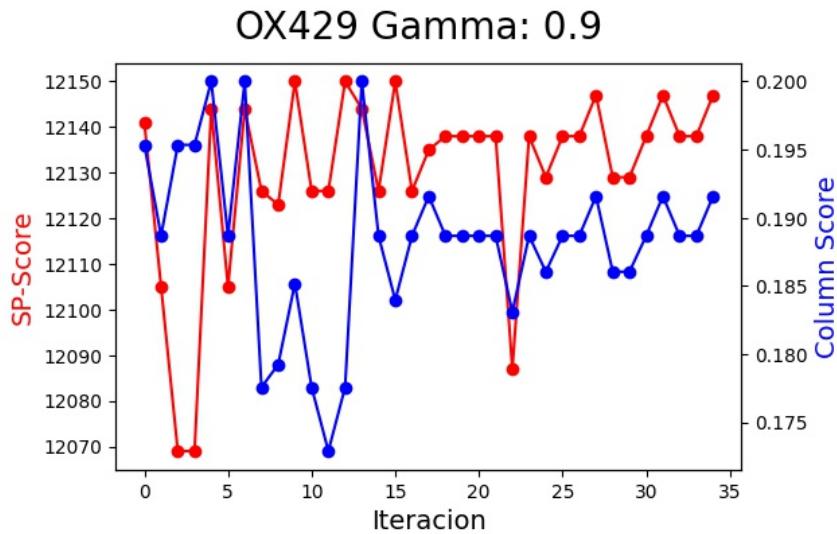


Figura 4.8: [Permutaciones, OX429, $\gamma = 0.9$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de permutaciones. Se alcanza el máximo en el episodio 10.

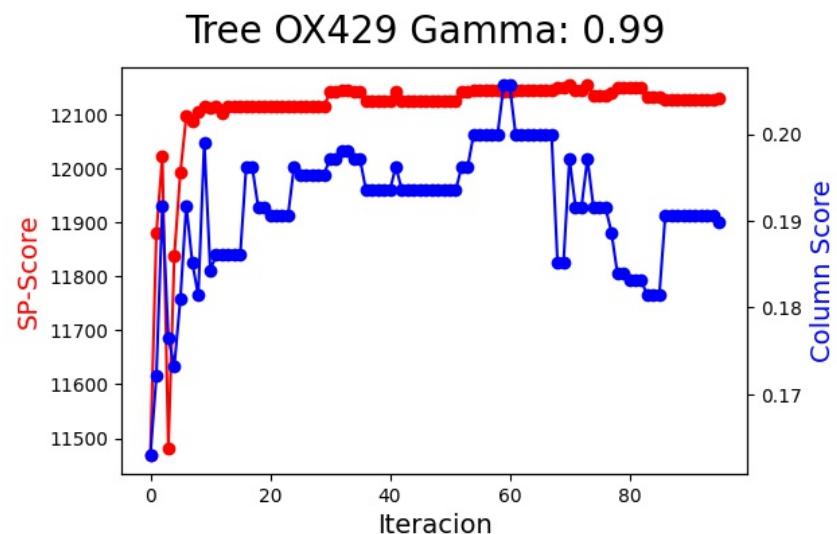


Figura 4.9: [Árboles, OX429, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de árboles. Se alcanza cierta estabilidad desde el episodio 10, presentándose mejoras esporádicas, hasta llegar al máximo en el episodio 70.

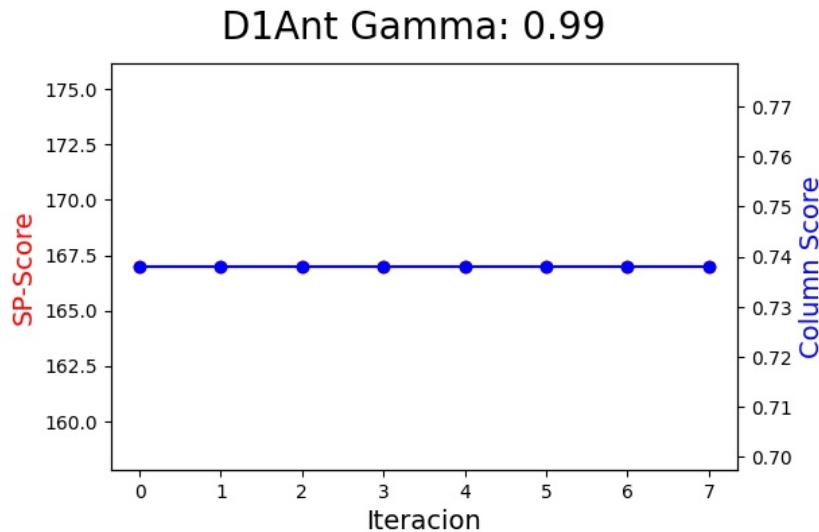


Figura 4.10: [Permutaciones, D1Ant, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de permutaciones. Se alcanza estabilidad desde el primer episodio.

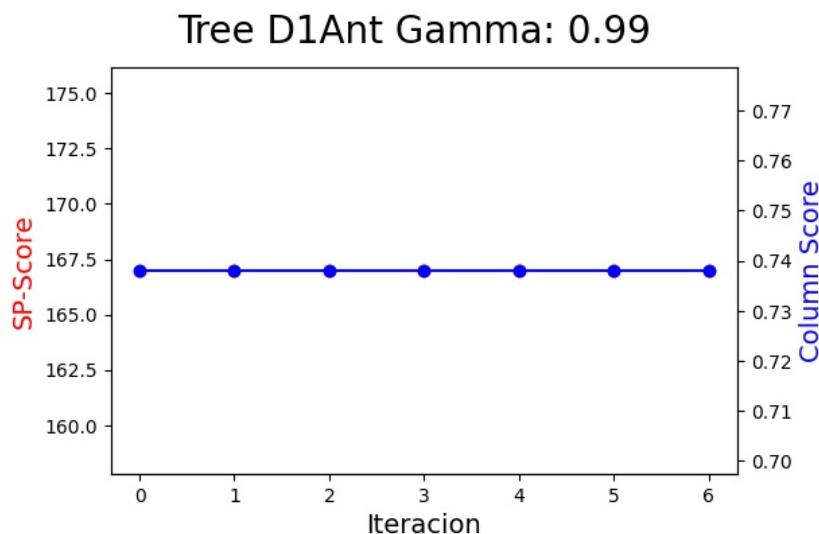


Figura 4.11: [Árboles, D1Ant, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de árboles. Se alcanza estabilidad desde el primer episodio.

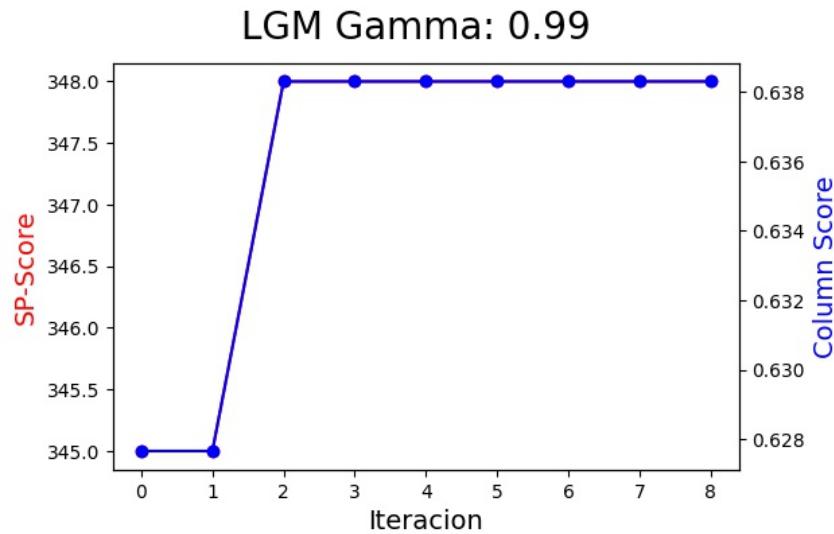


Figura 4.12: [Permutaciones, LGM, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de permutaciones. Se alcanza estabilidad en el tercer episodio.

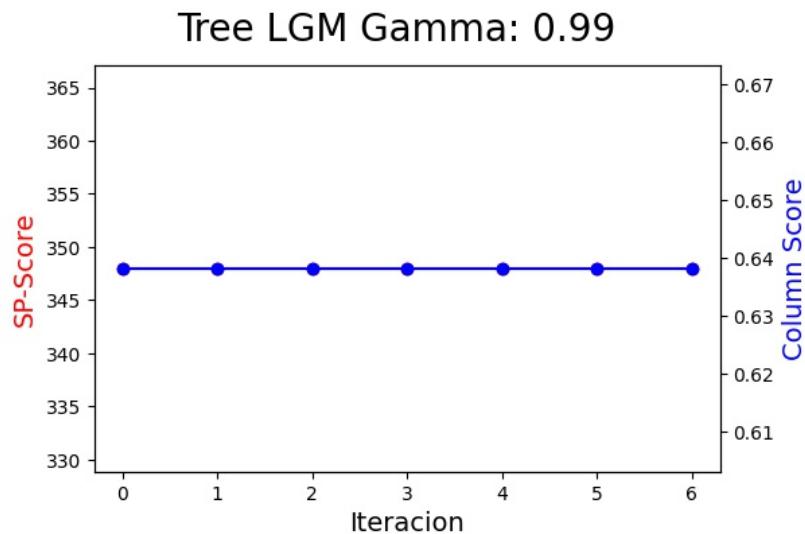


Figura 4.13: [Árboles, LGM, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de árboles. Se alcanza estabilidad desde el primer episodio.

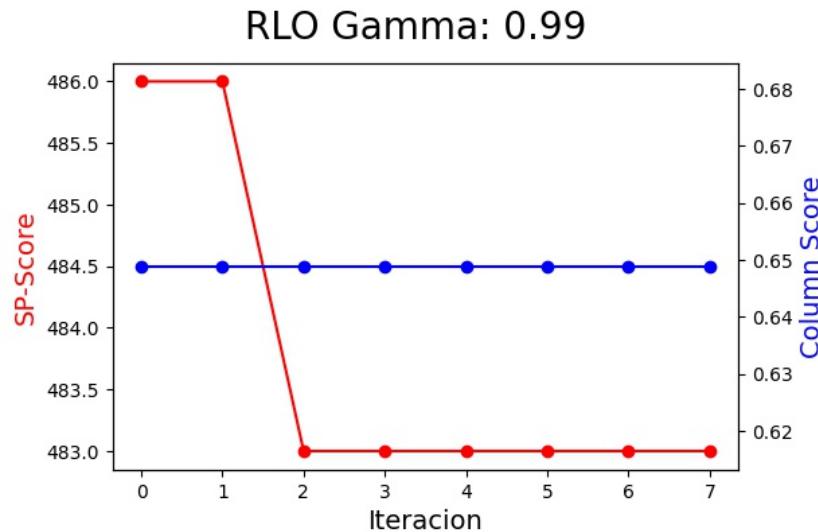


Figura 4.14: [Permutaciones, RLO, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de permutaciones. Se alcanza el máximo en el primer episodio, posteriormente se alcanza estabilidad sobre un SP score menor.

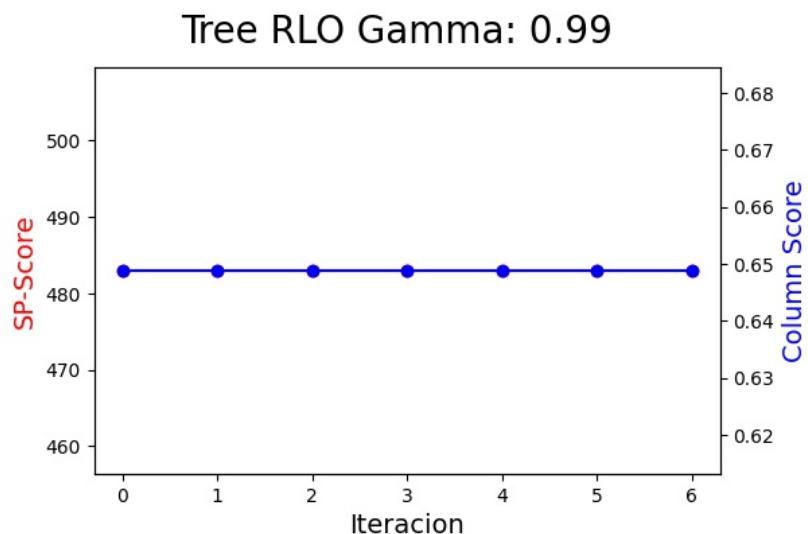


Figura 4.15: [Árboles, RLO, $\gamma = 0.99$]. Evolución del SP-score y del column score conforme los episodios realizados bajo el espacio de árboles. Se alcanza estabilidad desde el primer episodio.

Data Set	RL-Permutation					RL-Tree					Tiempo (seg)	
	SP	AL	EM	CS	γ	Tiempo (seg)	SP	AL	EM	CS	γ	
HC	18627	212	198	0.93	Todos	45-50	18627	212	198	0.93	Todos	45-50
PA	18884	1100	909	0.83	Todos	19-20	18884	1100	909	0.83	0.01 y 0.1	19-20
469OX	660	352	150	0.43	Todos	4-5	681	359	156	0.43	Todos	4-5
429OX	12150	216	40	0.19	0.9	106	12153	213	42	0.20	0.99	378
DIAnt	167	42	31	0.74	Todos	3-5	167	42	31	0.74	Todos	3-5
LGM	348	94	60	0.64	Todos	3-5	348	94	60	0.64	Todos	3-5
RLO	486	131	85	0.65	Todos	3-5	483	131	85	0.65	Todos	3-5

Tabla 4.2: Resultados de las métricas SP,AL,EM,CS en los 7 conjuntos de los métodos propuestos, así como los valores de γ en los cuales se dio el máximo valor del SP-score y el tiempo aproximado de ejecución que tomo llegar al óptimo.

Data Set	RL-Permutation				RL-Tree				CLUSTALW				MAFFT			
	SP	AL	EM	CS	SP	AL	EM	CS	SP	AL	EM	CS	SP	AL	EM	CS
HC	18627	212	198	0.93	18627	212	198	0.93	18627	212	198	0.93	18627	212	198	0.93
PA	18884	1100	909	0.83	18884	1100	909	0.83	18827	1098	905	0.82	18860	1098	907	0.83
469OX	660	352	150	0.43	681	359	156	0.43	464	359	134	0.37	549	342	122	0.36
429OX	12150	216	40	0.19	12153	213	42	0.20	9575	186	35	0.19	10218	183	35	0.19
DI.Ant	167	42	31	0.74	167	42	31	0.74	149	42	28	0.67	134	41	25	0.61
LGM	348	94	60	0.64	348	94	60	0.64	345	94	59	0.63	345	94	59	0.63
RLO	486	131	85	0.65	483	131	85	0.65	480	130	83	0.64	471	129	81	0.63

Tabla 4.3: Resultados de las métricas SP, AL, EM, CS en los 7 conjuntos de los métodos propuestos y de CLUSTAL y MAFFT.

Data Set	RL-Permutation					RL-Tree					Mircea - 2016					Jafari - 2019				
	SP	AL	EM	CS	SP	AL	EM	CS	SP	AL	EM	CS	SP	AL	EM	CS				
HC	18627	212	198	0.93	18627	212	198	0.93	18627	212	198	0.93	18627	212	198	0.93				
PA	18884	1100	909	0.83	18884	1100	909	0.83	18719	1110	918	0.83	18860	1110	918	0.83				
469OX	660	352	150	0.43	681	359	156	0.43	565	378	171	0.45	565	378	171	0.45				
429OX	12150	216	40	0.19	12153	213	42	0.20	8668	205	43	0.21	10218	206	43	0.21				
D1	167	42	31	0.74	167	42	31	0.74	167	42	31	0.74	-	-	-	-				
LGM	348	94	60	0.64	348	94	60	0.64	345	99	66	0.67	345	99	66	0.67				
RL0	486	131	85	0.65	483	131	85	0.65	486	133	87	0.65	486	138	87	0.63				

Tabla 4.4: Resultados de las métricas SP, AL, EM, CS en los 7 conjuntos de los métodos propuestos y los obtenidos por Mircea y cols. (2016) y Jafari y cols. (2019).

4.2.3. Comparación OXBENCH

Con el objetivo de distinguir entre el rendimiento del enfoque de permutaciones y el método de árboles se realizó una comparación entre los métodos utilizando algunos de los conjuntos presentes en **oxbench_mdsa_all**, comparando los alineamientos producidos por los enfoques propuestos con el enfoque proporcionado en dicha base de datos. Cabe destacar que debido a que no se cuenta con los alineamientos ni métodos exactos para reproducir los resultados de [Mircea y cols. \(2014\)](#), [Mircea y cols. \(2016\)](#) y [Jafari y cols. \(2019\)](#), no se aplican dichos métodos sobre los conjuntos que fueron utilizados. De los 672 archivos presentes en **oxbench_mdsa_all** se eligieron aquellos conjuntos que contuvieran entre 4 y 12 secuencias y que todas sus secuencias de nucleótidos no tuvieran caracteres distintos a los nucleótidos del ADN ². De esta forma fueron seleccionados 189 conjuntos, sobre los cuales se aplicó el algoritmo Deep Q-Learning with experience replay con el mismo esquema de hiperparámetros de la subsección 4.2.1, con la adición de que fue fijado el valor de gamma en 0.99 pues en los experimentos anteriores un factor de descuento cercano a uno (con mayor visión hacia adelante) arrojó mejores resultados.

Así, fueron obtenidos dos alineamientos para cada archivo, uno con el enfoque de permutaciones (denotado por RL-Permutation) y otro con el enfoque de árboles (denotado por RL-Tree). Además, se cuenta con el alineamiento proporcionado por [Carroll y cols. \(2007\)](#), obtenido usando como base los alineamientos de proteínas de [Raghava, Searle, Audley, Barber, y Barton \(2003\)](#). En la Tabla A del Anexo A podemos encontrar para cada uno de los archivos las métricas SP-score, AL, EM, CS para los 3 alineamientos obtenidos. En la Tabla A.2 del Anexo A, se encuentra la diferencia en el SP-score y el column score de los alineamientos obtenidos por los métodos propuestos y el alineamiento original. También en dicha tabla se encuentra información adicional relativa la cantidad de secuencias e información sobre sus longitudes para cada uno de los archivos.

²Algunos archivos de secuencias contienen caracteres que denotan incertidumbre sobre el verdadero nucleótido presente en la secuencia.

Considerando que la cantidad de secuencias a alinear es determinante en la complejidad del problema, se resumen los resultados anteriores agrupados por cantidad de secuencias a alinear. Como se ve en la Tabla 4.5, en todos los casos con los enfoques propuestos en promedio, se tiene una mejoría notable tanto en el SP-score como en el column score. Consideremos ahora la proporción de mejora obtenido para cada conjunto de secuencias de los archivos. Debido a que los métodos utilizados tienen como objetivo maximizar el SP-score específico utilizado, es de esperar que en la Figura 4.16 se tengan mejoras importantes del SP-score. Sin embargo, debemos destacar que el column Score también mostró mejoría en forma importante en todos los casos, como podemos ver en la Figura 4.17.

Ambos métodos mejoran el baseline del alineamiento proveído por Carroll y cols. (2007) en forma similar. De especial interés resulta analizar el grupo con 11 secuencias, pues los porcentajes de mejora para dicho caso son evidentemente mayores, esto se realizará en la siguiente sección.

Resultados utilizando scoring distintos.

Utilizando los 189 conjuntos de secuencias de OXBENCH empleadas anteriormente, se realizaron los mismos experimentos utilizando los siguientes scorings:

- **Scoring 1:** +2 para nucleótidos iguales alineados, -1 para nucleótidos distintos alineados y -4 para cuando un nucleótido es alineado con un gap. En este scoring se da una mayor penalización a los gaps.
- **Scoring 2:** +4 para nucleótidos iguales alineados, -1 para nucleótidos distintos alineados y -2 para cuando un nucleótido es alineado con un gap. En este scoring se da una mayor recompensa al alineamiento de nucleótidos iguales.
- **Scoring 3:** +2 para nucleótidos iguales alineados, -3 para nucleótidos distintos alineados y -2 para cuando un nucleótido es alineado con un gap. En este scoring

se penaliza más el alineamiento de nucleótidos distintos que a la colocación de gaps.

Los resultados resumidos por cantidad de secuencias de la mejora del SP-score y del column score de cada scoring están en la Figura 4.18, la Figura 4.19 y la Figura 4.20. Resulta interesante que bajo los 3 scoring's los métodos propuestos mejoran el column score con efectos relativos similares según el grupo de cantidad de secuencias. En cuanto al SP-score, bajo el scoring 1, que penaliza más severamente la presencia de gaps, se tienen mejoras en casi todos los grupos a excepción del grupo conjuntos con 11 secuencias; bajo el scoring 2 se tuvieron mejoras en todos los grupos; y, finalmente, bajo el scoring 3 ningún grupo, a excepción del grupo de conjuntos con 11 secuencias presento, mejoría. De lo anterior, podemos observar que los métodos propuestos generan alineamientos con un buen column score, independientemente del scoring empleado. En cuanto a la comparación de los enfoques, podemos notar que no presentan diferencias notables en la calidad de los alineamientos generados, esto aunado a que el enfoque de permutaciones es más rápido, indica que dicho enfoque se pueda considerar como una mejor opción que utilizar el enfoque de árboles.

N	SPOrig	ALOrig	EMOrig	CSOrig	SPPer	ALPer	EMPer	CSPer	SPTree	ALTtree	EMTree	CSTree	Lmean	Lmax	Lmin	Archivos
4	1492.457	474.857	152.800	0.326	2091.514	486.057	201.029	0.415	2090.529	484.829	200.286	0.414	433.125	458.657	413.100	70
5	2193.786	480.786	133.095	0.321	3356.571	484.524	178.952	0.400	3337.143	484.643	177.905	0.398	421.214	450.286	394.000	42
6	3734.778	455.444	111.259	0.258	4913.556	469.000	151.926	0.330	4886.444	469.370	151.593	0.329	413.000	430.556	392.222	27
7	3968.800	462.600	95.600	0.237	5753.600	477.667	128.333	0.297	5787.800	479.000	127.200	0.293	398.257	429.000	376.600	15
8	4206.500	382.500	82.333	0.302	6082.500	403.167	102.500	0.343	6151.000	406.000	102.500	0.340	328.000	347.500	302.500	6
9	6000.600	422.100	57.700	0.149	7353.300	463.900	79.500	0.186	7556.400	463.200	81.000	0.189	374.433	399.000	354.300	10
10	8705.000	362.500	80.167	0.213	10093.500	377.000	91.000	0.238	10061.000	381.833	92.000	0.239	305.450	342.000	286.500	6
11	6270.333	550.000	54.111	0.114	12692.667	566.667	85.889	0.165	12602.667	565.111	83.778	0.161	429.606	498.000	387.333	9
12	17457.750	366.750	76.500	0.212	19797.000	399.500	99.000	0.254	19689.750	401.000	97.750	0.251	339.438	351.750	324.750	4

Tabla 4.5: Métricas promedio de los tres alineamientos agrupados por cantidad de secuencias. N es la cantidad de secuencias. Archivos es la cantidad de conjuntos de secuencias con las que se cuenta con la cantidad de secuencias indicadas por N. Lmean, Lmax, Lmin denota el promedio sobre los correspondientes archivos para la longitud media, longitud máxima y longitud mínima de las secuencias a alinear.

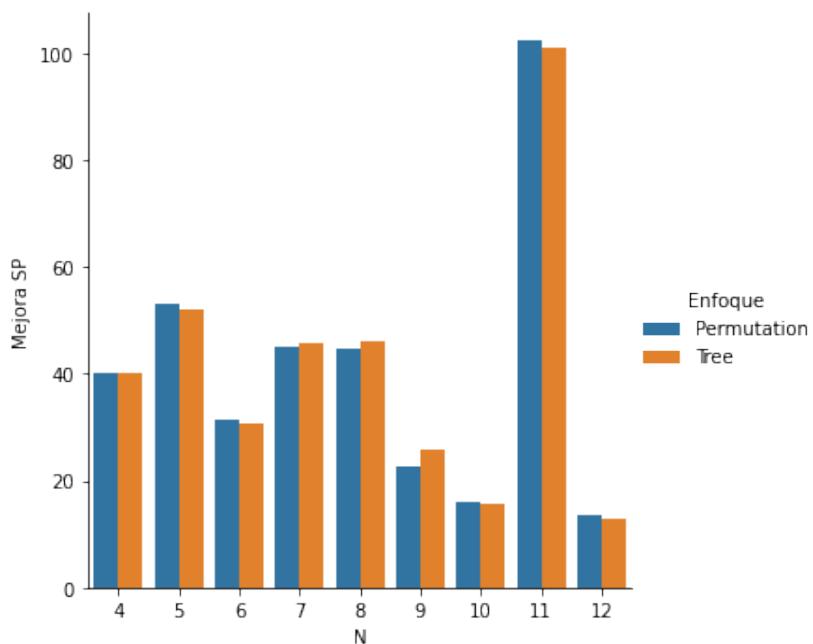


Figura 4.16: Porcentaje de mejora del SP-score respecto al alineamiento original para los enfoques de permutaciones y de árboles.

4.3. Métrica GUIDANCE

Al realizar un alineamiento progresivo, es necesario tener un árbol guía como base. Al ser desconocida la filogenia original de la que provienen los datos, se introducen niveles de incertidumbre sobre el alineamiento final. Penn y cols. (2010) desarrollaron la métrica GUIDANCE (guide tree-based alignment confidence) que considera la robustez de los alineamientos obtenidos con respecto al árbol guía utilizado. El procedimiento para calcular la métrica para una alineamiento base MSA_0 es el siguiente:

1. Generar una muestra de 100 alineamientos bootstrap siguiendo el procedimiento de (Felsenstein, 1985). Cada alineamiento se obtiene muestreando con reemplazo aleatoriamente columnas de MSA_0 hasta completar un alineamiento con la misma longitud que MSA_0 .
2. Para cada uno de los alineamientos, se obtiene una matriz de distancias utilizando el SP-score entre cada par de secuencias (ver Pevsner (2015)). Posteriormente, usando la matriz de distancias para cada caso, se genera un árbol guía utilizando el algoritmo Neighbor-Joining de Saitou y Nei (1987). De esta for-

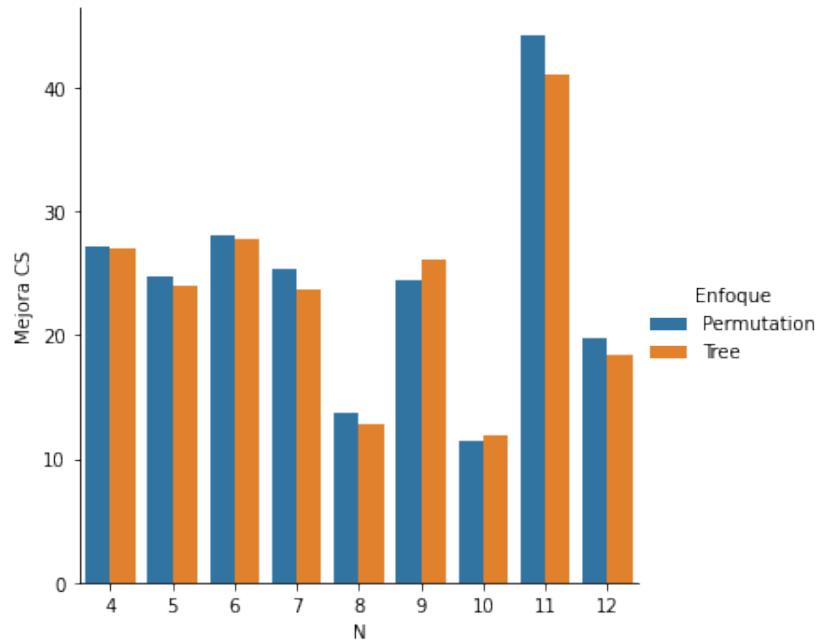


Figura 4.17: Porcentaje de mejora del column score respecto al alineamiento original para los enfoques de permutaciones y de árboles.

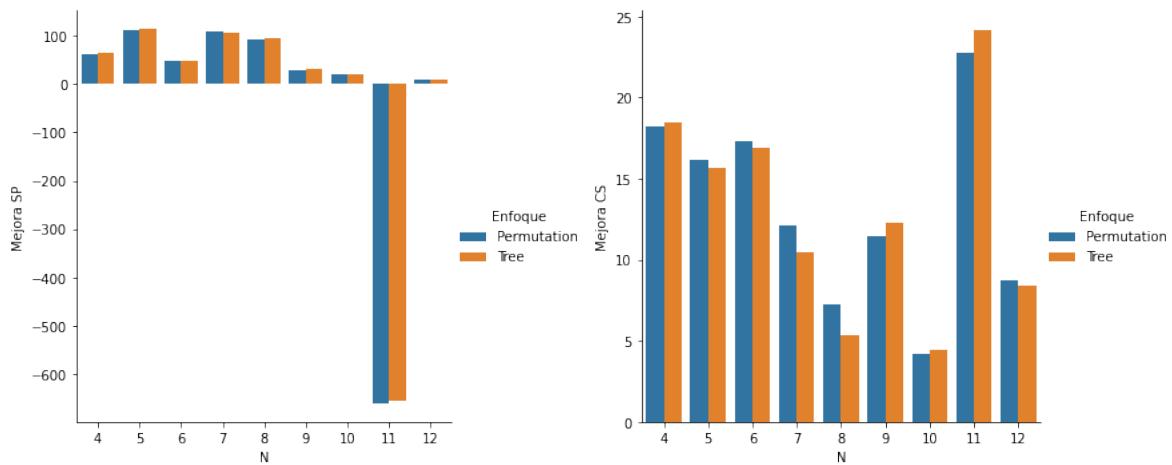


Figura 4.18: Mejora relativa del SP-score y del column score utilizando el scoring 1 con el enfoque de permutaciones y el enfoque de árboles.

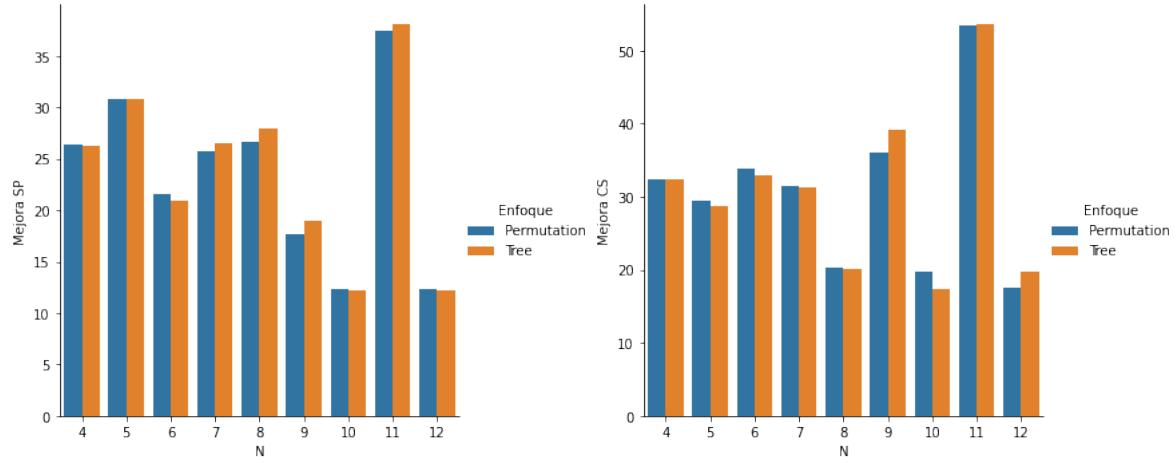
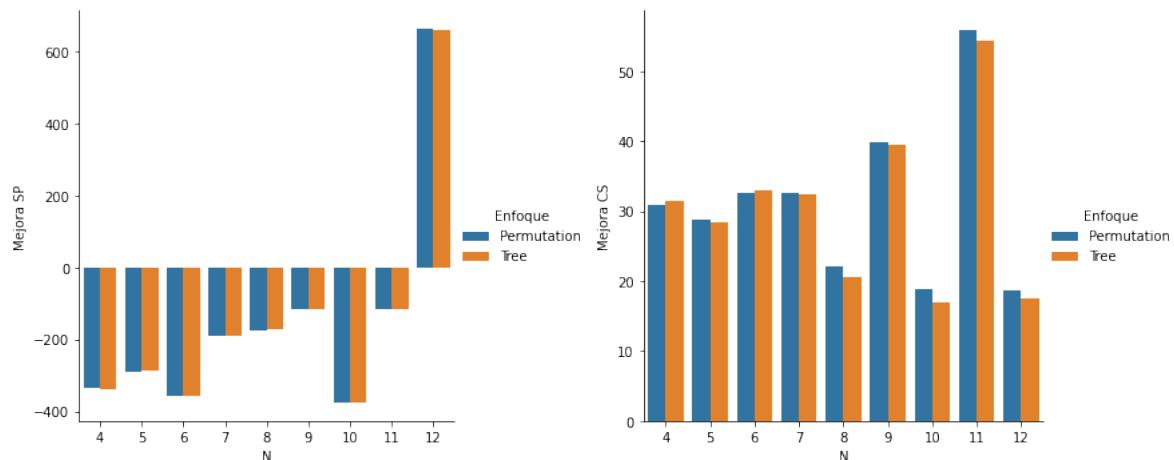


Figura 4.19: Mejora relativa del SP-score y del column score utilizando el scoring 2 con el enfoque de permutaciones y el enfoque de árboles.



ma se tienen 100 árboles $TREE_1, \dots, TREE_{100}$. Cabe mencionar que se toman 100 árboles pues es la cantidad utilizada por Penn y cols. (2010). Sin embargo, incrementar la cantidad de muestras bootstrap se pueden obtener scores más precisos.

3. Se realiza alineamiento progresivo utilizando los árboles $TREE_1, \dots, TREE_{100}$ para obtener MSA_1, \dots, MSA_{100} , respectivamente.
4. Se calculan los GUIDANCE score para cada columna de la siguiente forma:
 - a) Si en la columna está presente sólo un nucleótido, su score es 0.
 - b) Si en la columna están presentes m nucleótidos, con $m > 1$, para cada pareja de nucleótidos (cada nucleótido siempre es considerado como un elemento indexado de cada secuencia para cada secuencia) se realiza un conteo de en cuántos alineamientos dicha pareja fue también colocada en una misma columna. Sea C el resultado de sumar los conteos sobre todas las parejas presentes, entonces se tiene que $C \leq 100 \times \binom{m}{2}$, pues en un caso extremo, la columna en cuestión aparece en forma idéntica en los 100 alineamientos generados. Finalmente se asigna como GUIDANCE score a la cantidad $\frac{C}{100 \times \binom{m}{2}}$.
5. Se calculan los GUIDANCE score para cada nucleótido de MSA_0 :
 - a) Si el nucleótido en cuestión está acompañado solo con gaps, se le asigna un score de cero.
 - b) Si el nucleótido esta alineado con otros $r > 0$ nucleótidos, se realiza un conteo de en cuantos alineamientos las r parejas aparecen en los 100 alineamientos generados. Sea R dicha cantidad; es fácil ver que $R < 100 \times r$. El score asignado es $\frac{R}{100 \times r}$.

En la Figura 4.21 podemos ver una esquematización de los pasos anteriores. Los GUIDANCE score son siempre números entre 0 y 1, que denotan el nivel de confianza ante la incertidumbre que el árbol guía provoca. Un score de 0 significa que en los 100

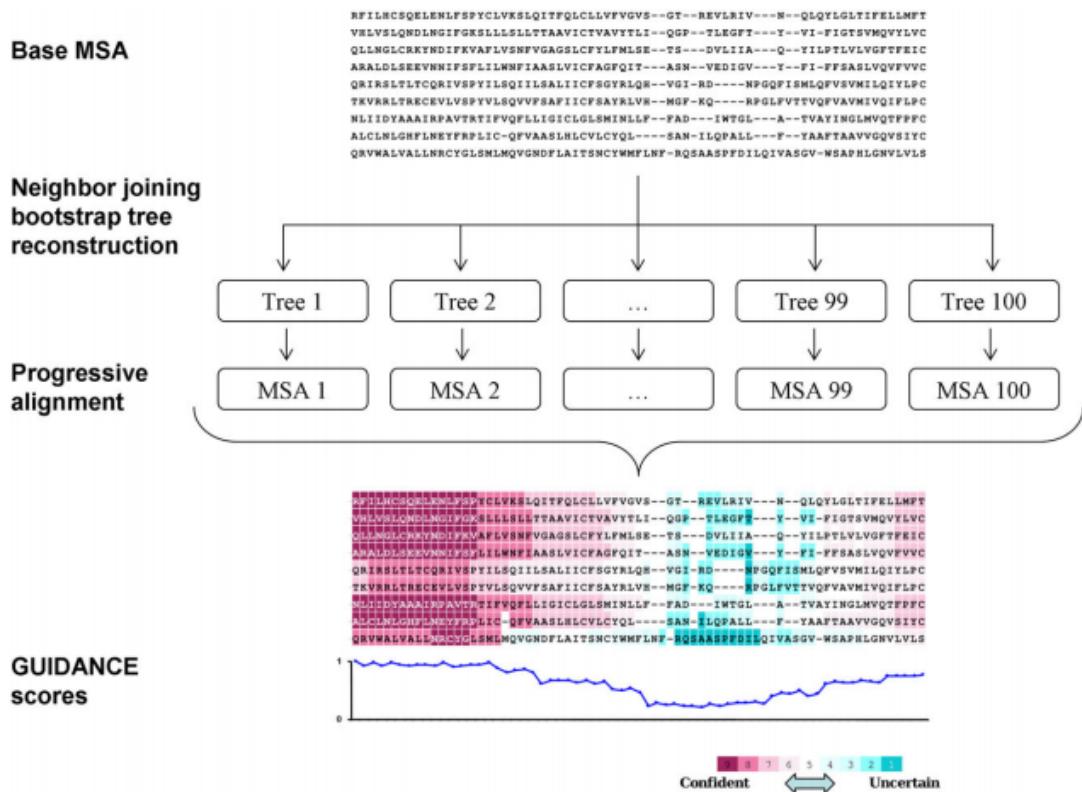


Figura 4.21: Esquematización de los pasos necesario para obtener el GUIDANCE score. La figura fue tomada de Penn y cols. (2010), donde se consideran secuencias de aminoácidos.

alineamientos generados no se encontraron las asociaciones descritas por la columna en cuestión. Mientras que un score de 1 implica que en todos los casos se mantuvieron las mismas asociaciones.

Obteniendo los GUIDANCE score para los conjuntos con $N = 11$ secuencias de la sección anterior podemos realizar una comparación adicional al rendimiento de los métodos con respecto al alineamiento proveido por Carroll y cols. (2007). En la Figura 4.22 podemos notar que en 7 de los 9 casos, uno de los dos alineamiento obtuvo un GUIDANCE score promedio mayor.

En el caso del archivo 86t10, ambos métodos obtienen un mejor nivel de confianza; también podemos notar en la Figura 4.23 que los enfoques propuestos obtienen un acomodo más confiable de la onceava secuencia. Los resultados para el archivo 136t21,

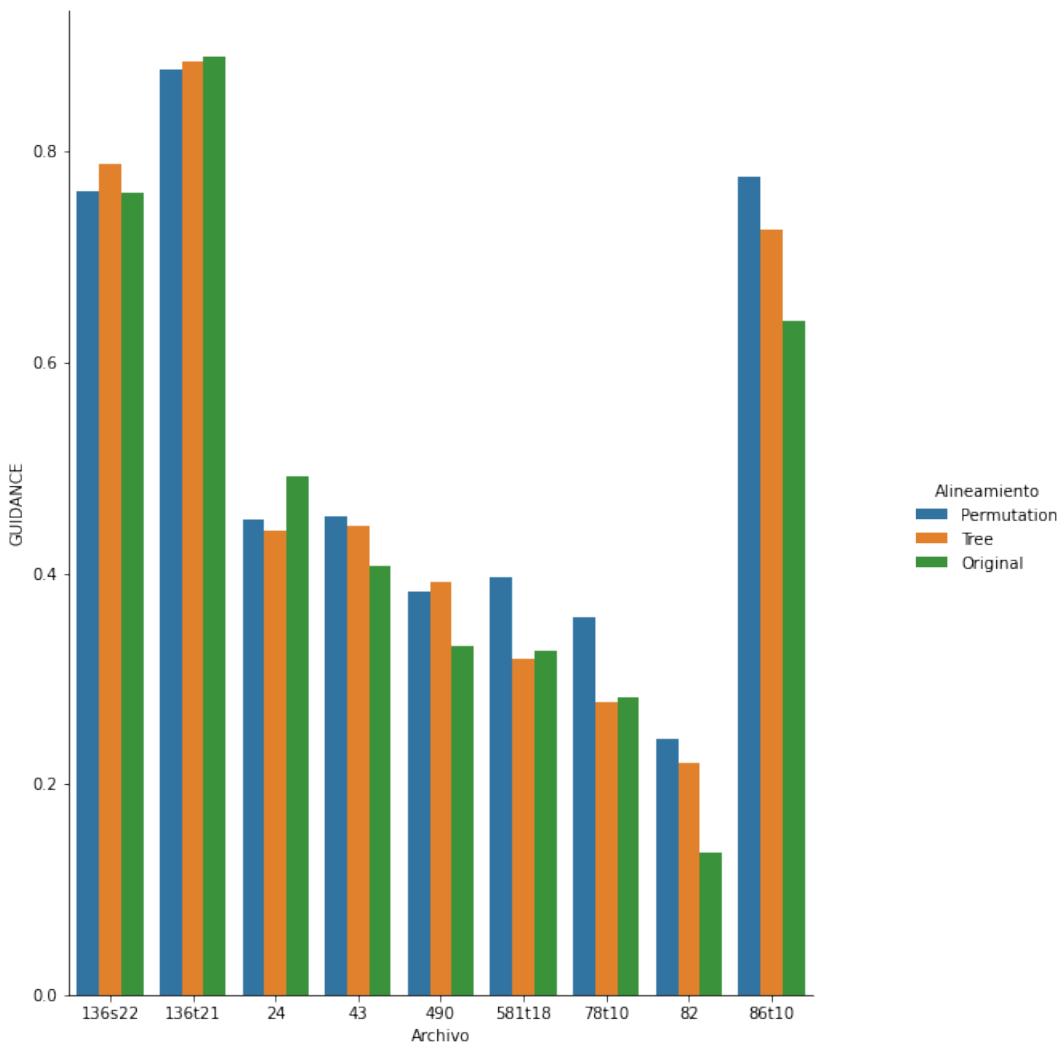


Figura 4.22: Promedio de los GUIDANCE score de las columnas para cada uno de los alineamientos.

donde la confianza fue mayor para el alineamiento original por un pequeño margen, muestran en la figura 4.24 que los enfoques propuestos mantuvieron las zonas de confianza. En particular, con el enfoque de árboles las últimas dos secuencias fueron alineadas con mayor confianza. Finalmente, revisando el caso del archivo 24, donde también el alineamiento original es más confiable por un margen mayor, podemos observar en la Figura 4.25 que si bien se tienen un patrón menos claro de las zonas de confianza, del lado derecho (formado con las últimas columnas) se obtienen zonas de confianza que involucran a todas las secuencias, a diferencia del alineamiento original.

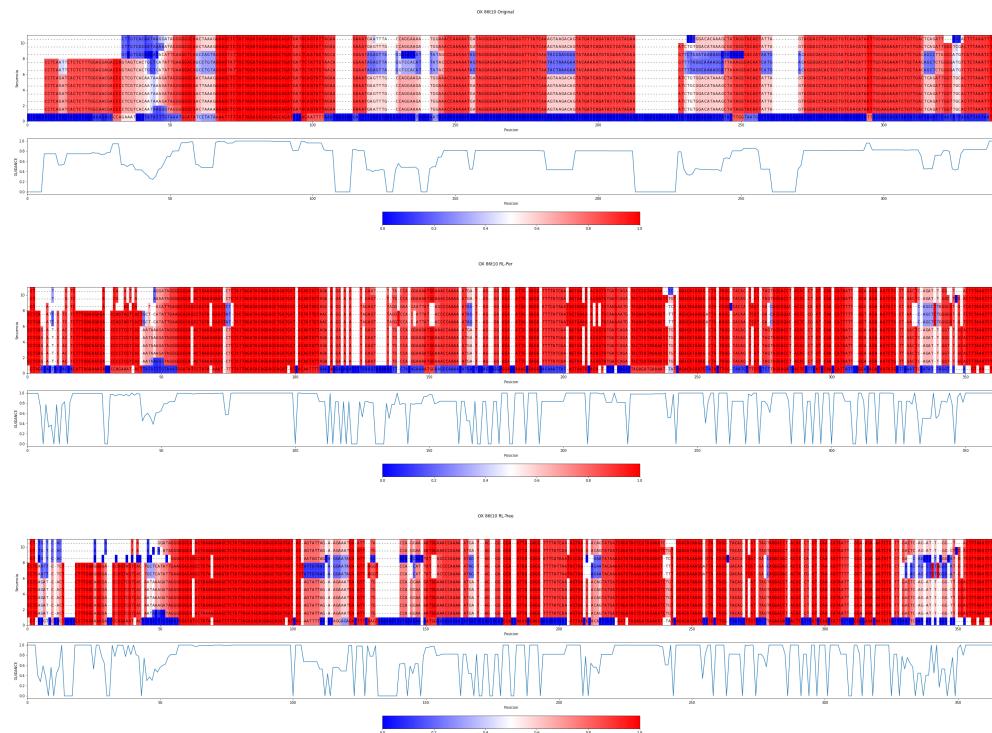


Figura 4.23: GUIDANCE scores del alineamiento original, el obtenido con el enfoque de permutaciones y el obtenido con el enfoque de árboles del archivo 86t10, respectivamente.

4.4. NFQ

El enfoque de los sufijos presentó resultados poco satisfactorios al usar el algoritmo Deep Q-Learning with experience replay; después de cientos de iteraciones, los alineamientos producidos eran de una calidad muy baja. Aprovechando que es posible simular experiencias de este enfoque sin necesidad de simular episodios enteros, se optó por usar el algoritmo Neural Fitted Q Iteration visto en el Capítulo 2 sobre el conjunto de datos mostrados en la Figura 4.26.

Para aproximar Q se utilizó el mismo esquema de la Figura 4.1. Se realizaron 4 alineamientos múltiples, con las primeras 2, 3, 4 y 5 secuencias en la Figura 4.26, esto con la finalidad de ver el desempeño del método conforme la cantidad de secuencias se incrementa. Para este algoritmo se eligieron 1500 iteraciones, una γ de 0.99, una paciencia de 1000 iteraciones sin mejora en el SP-score, con el mismo scoring de la Sección 4.2, y cada 50 iteraciones se realizó un remuestreo del batch de experiencias.

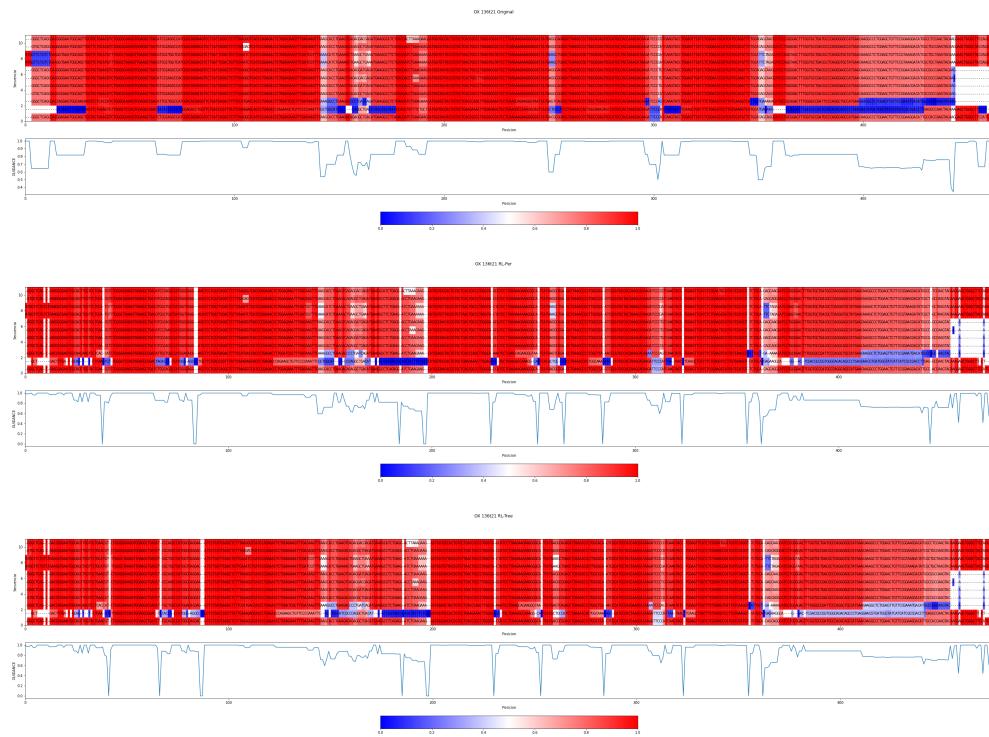


Figura 4.24: GUIDANCE scores del alineamiento original, el obtenido con el enfoque de permutaciones y el obtenido con el enfoque de árboles del archivo 136t21, respectivamente.

Los resultados del historial de entrenamiento se encuentra en la Figura 4.27 donde podemos observar que, a mayor cantidad de secuencias, más errático es el entrenamiento.

La Tabla 4.6 contiene las métricas de calidad de los alineamientos obtenidos por los enfoques de permutaciones, árboles y sufijos. Podemos observar que para 4 o más secuencias, los alineamientos del método de sufijos son de muy baja calidad a pesar del gran tiempo de ejecución que requirieron. Se requieren realizar más estudios para aprovechar eficientemente el enfoque de sufijos. Es necesario encontrar formas más adecuadas de entrenar al método de sufijos para obtener mejores resultados; ese un trabajo que queda pendiente para un futuro cercano, como remarcamos en la sección de conclusiones.

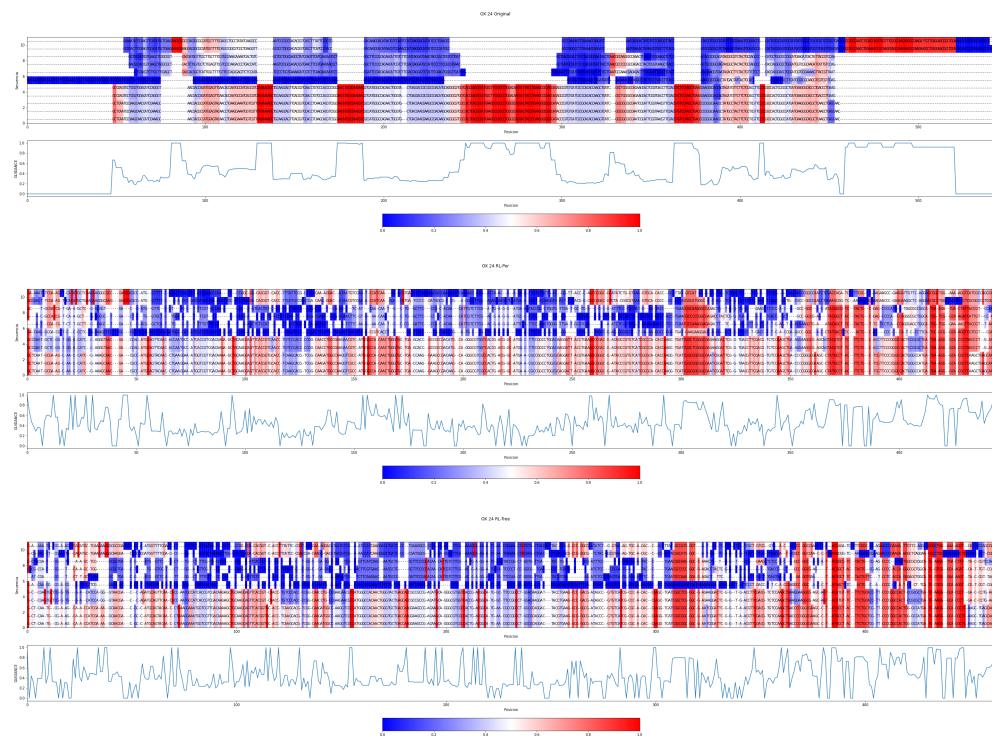


Figura 4.25: GUIDANCE scores del alineamiento original, el obtenido con el enfoque de permutaciones y el obtenido con el enfoque de árboles del archivo 24, respectivamente.

```

ATGGCGCCCCGA
GGTCATGGCGCCCCGA
GTCATGGCGCCCCGA
GTCATGGCGGTCCGAA
GTCATGGCTCCCCGAACC

```

Figura 4.26: 5 secuencias utilizadas para las pruebas con NFQ.

N	RL- Permutation					RL - Tree					RL - Suffix				
	SP	AL	EM	CS	T	SP	AL	EM	CS	T	SP	AL	EM	CS	T
2	16	16	12	0.75	2.66	16	16	12	0.75	2.73	16	16	12	0.75	369.22
3	62	16	12	0.75	4.20	62	16	12	0.75	3.30	56	16	11	0.68	492.92
4	114	17	10	0.58	5.59	114	17	10	0.58	4.93	36	17	3	0.17	441.34
5	182	19	9	0.47	8.02	182	19	9	0.47	6.78	20	18	1	0.05	749.28

Tabla 4.6: Resultados de las métricas usando los tres enfoques propuestos.

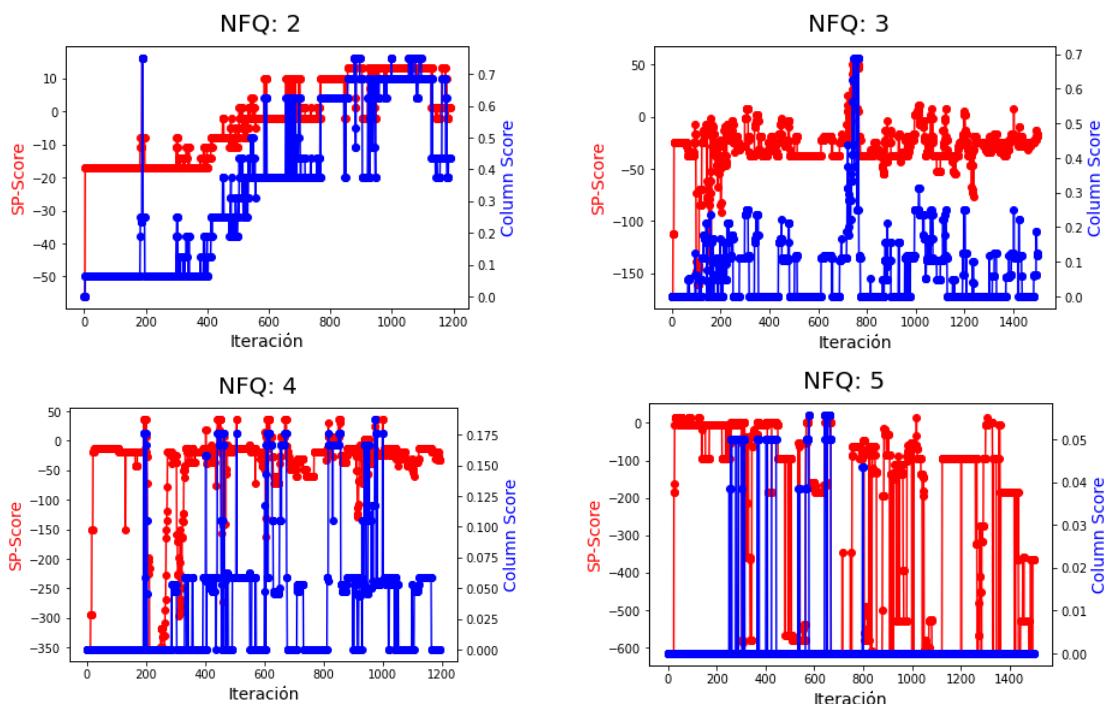


Figura 4.27: Historial del entrenamiento para los 4 casos. En el alineamiento de dos secuencias se observa como al transcurrir los cientos de episodios se incrementa el SP-score, algo similar sucede para 3 secuencias. En el alineamiento de 4 y 5 secuencias el comportamiento es mas errático.

Capítulo 5

Conclusiones y trabajo futuro

Los métodos basados en aprendizaje reforzado y en alineamiento progresivo funcionaron satisfactoriamente, en cuestión de segundos lograron dar alineamientos de buena calidad. Esto lo podemos constatar principalmente en la mejoría mostrada en el column score de los alineamientos de Carroll y cols. (2007), pues el column score es independiente del scoring empleado para crear los alineamientos. Sin embargo, aún es necesario realizar una exploración de otras técnicas de aprendizaje reforzado para mejorar el rendimiento de estos métodos así como una búsqueda sobre los parámetros fijados con la finalidad de obtener resultados aún mejores. Asimismo, es necesario realizar una exploración sobre el métodos de los sufijos, para determinar si es posible obtener soluciones que escapen del espacio de soluciones dadas por el alineamiento progresivo, a pesar de sacrificar la ganancia en cuanto a tiempo de cómputo.

Algunas posibilidades para el trabajo futuro contemplan el uso de cómputo en paralelo para la realización de episodios; utilizar técnicas de deep-learning para hacer un encoding de los estados del enfoque de sufijos para obtener representaciones interesantes y que sean más útiles en cuanto a que la información codificada pueda utilizarse eficientemente en algoritmos de aprendizaje reforzado; usar otras estrategias de exploración que involucren la duración de los episodios; y aplicar el enfoque de sufijos sobre particiones de las secuencias, resolviendo problemas de menor complejidad computacional, para posteriormente combinar las partes en un solo alineamiento.

De esta forma, considero que el aprendizaje reforzado puede ser una alternativa muy útil e interesante para el abordamiento de problemas combinatorios complejos que requieran una solución heurística.

Referencias

- Carroll, H., Beckstead, W., O'Connor, T., Ebbert, M., Clement, M., Snell, Q., y McClellan, D. (2007). Dna reference alignment benchmarks based on tertiary structure of encoded proteins. *Bioinformatics*, 23(19), 2648–2649.
- Choudhary, A. (2020, Apr). *Deep q-learning: An introduction to deep reinforcement learning*. Descargado de <https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., y Stein, C. (2009). *Introduction to algorithms*. MIT press.
- Czibula, I.-G., Czibula, G., y Bocicor, M.-I. (2011). A software framework for solving combinatorial optimization tasks. *Studia Universitatis Babes-Bolyai, Informatica*, 56(3).
- Documentación de la librería cppyy. (2020). Descargado de <https://cppyy.readthedocs.io/en/latest/index.html>
- Edgar, R. C. (2004a). Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic acids research*, 32(1), 380–385.
- Edgar, R. C. (2004b). Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5), 1792–1797.
- Ewens, W. J., y Grant, G. R. (2001). Statistical methods in bioinformatics: an introduction.
- Felsenstein, J. (1985). Confidence limits on phylogenies: an approach using the bootstrap. *evolution*, 39(4), 783–791.
- Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT Press. (<http://>

- www.deeplearningbook.org
- Google colab. (2020). Descargado de <https://colab.research.google.com/notebooks/intro.ipynb>
- Higgins, D. G., y Sharp, P. M. (1988). Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1), 237–244.
- Isaev, A. (2006). *Introduction to mathematical methods in bioinformatics*. Springer.
- Jafari, R., Javidi, M. M., y Rafsanjani, M. K. (2019). Using deep reinforcement learning approach for solving the multiple sequence alignment problem. *SN Applied Sciences*, 1(6), 592.
- Kanz, C., Aldebert, P., Althorpe, N., Baker, W., Baldwin, A., Bates, K., ... others (2005). The embl nucleotide sequence database. *Nucleic acids research*, 33(suppl_1), D29–D33.
- Katoh, K., Misawa, K., Kuma, K.-i., y Miyata, T. (2002). Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic acids research*, 30(14), 3059–3066.
- Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., y Pérez, P. (2020). Deep reinforcement learning for autonomous driving: A survey. *arXiv preprint arXiv:2002.00444*.
- Knuth, D. E. (1997). *The art of computer programming* (Vol. 3). Pearson Education.
- Lange, S., Gabel, T., y Riedmiller, M. (2012). Batch reinforcement learning. En *Reinforcement learning* (pp. 45–73). Springer.
- Mircea, I.-G., Bocicor, I., y Czibula, G. (2016). A reinforcement learning based approach to multiple sequence alignment. En *International workshop soft computing applications* (pp. 54–70).
- Mircea, I.-G., Bocicor, I., y Dîncu, A. (2014). On reinforcement learning based multiple sequence alignment. *Studia Universitatis Babes-Bolyai, Informatica*, 59(2).
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

- Notredame, C., Higgins, D. G., y Heringa, J. (2000). T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of molecular biology*, 302(1), 205–217.
- Penn, O., Privman, E., Landan, G., Graur, D., y Pupko, T. (2010). An alignment confidence score capturing robustness to guide tree uncertainty. *Molecular biology and evolution*, 27(8), 1759–1767.
- Pevsner, J. (2015). *Bioinformatics and functional genomics*. John Wiley & Sons.
- Polanski, A., y Kimmel, M. (2007). Sequence alignment. *Bioinformatics*, 155–185.
- Raghava, G., Searle, S. M., Audley, P. C., Barber, J. D., y Barton, G. J. (2003). Oxbench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC bioinformatics*, 4(1), 47.
- Ramakrishnan, R. K. (2018). *Exploring reinforcement learning techniques in multiple sequence alignment* (Tesis Doctoral no publicada). McGill University Libraries.
- Ramakrishnan, R. K., Singh, J., y Blanchette, M. (2018). Rlalign: A reinforcement learning approach for multiple sequence alignment. En *2018 ieee 18th international conference on bioinformatics and bioengineering (bibe)* (pp. 61–66).
- Saitou, N., y Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4), 406–425.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... others (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419), 1140–1144.
- Sutton, R. S. (1998). Reinforcement learning: Past, present and future. En *Asia-pacific conference on simulated evolution and learning* (pp. 195–197).
- Who. (2020). Descargado de <https://www.who.int/emergencies/diseases/novel-coronavirus-2019/events-as-they-happen>
- Wiering, M., y Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, 12, 3.
- Xiang, X., Zhang, D., Qin, J., y Fu, Y. (2010). Ant colony with genetic algorithm based on planar graph for multiple sequence alignment. *Information Technology Journal*, 9(2), 274–281.

- Yin, C. (2020). Genotyping coronavirus sars-cov-2: methods and implications. *Genomics*.
- Zhang, T., Wu, Q., y Zhang, Z. (2020). Probable pangolin origin of sars-cov-2 associated with the covid-19 outbreak. *Current Biology*.

Apéndice A

Resultados OXBENCH

En las siguientes páginas, se presentan un conjunto de tablas que comparan los resultados del alineamiento múltiple generado, para los conjuntos de datos de la base de OXBENCH, por los enfoques de árboles, permutaciones y los alineamiento múltiples ya proporcionados en la base de datos. Si bien los alineamientos ya proporcionados fueron producidos bajo criterios no necesariamente comparables con los enfoques basados en SP-scores, los resultados arrojan luz de que tan buenos son para reflejar la estructura de proteínas.

Archivo	SP-Score	Original			RL-Permutation			RL-Tree				
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM	
104s10	2136	222	93	0.418918919	2154	228	98	0.429825	2154	228	98	0.429825
104s13	1284	222	100	0.45045045	1338	233	111	0.476395	1338	233	111	0.476395
104s14	3432	219	86	0.392694064	3462	230	90	0.391304	3417	226	90	0.39823
104s16	1824	222	75	0.337837838	1890	238	89	0.37395	1863	237	84	0.35443
104s7	1878	219	143	0.652968037	1887	221	145	0.656109	1887	221	145	0.656109
104t18	2409	258	102	0.395348837	2502	270	114	0.422222	2508	273	116	0.424908
10s10	7728	507	66	0.130177515	9117	544	84	0.154412	10068	535	82	0.153271
10s3	3609	489	285	0.582822086	3870	492	312	0.634146	3864	492	311	0.632114
10t8	5385	489	226	0.462167689	5691	496	249	0.502016	5700	495	249	0.50303
10t9	1407	456	135	0.296052632	1872	494	188	0.380567	1875	493	189	0.383367
113	-714	495	62	0.125252525	651	480	131	0.272917	576	491	129	0.262729
115	177	480	78	0.1625	1131	495	140	0.282828	1152	489	138	0.282209
116	-873	696	62	0.08908046	2280	684	174	0.254386	2238	683	172	0.25183
118	669	213	65	0.305164319	795	221	82	0.371041	825	220	84	0.381818
120	501	285	66	0.231578947	687	308	93	0.301948	687	307	85	0.276873
121	240	597	84	0.140703518	1644	607	185	0.304778	1674	600	185	0.308333
123	-1839	462	14	0.03030303	444	469	66	0.140725	417	488	73	0.14959

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree			
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM
12s55	8448	345	171	0.495652174	8550	348	176	0.505747	8547	350	176
12s58	3213	333	262	0.786786787	3213	333	262	0.786787	3213	333	262
12s59	7554	327	229	0.70030581	7749	328	236	0.719512	7767	328	237
12s81	2844	324	215	0.663580247	2871	326	220	0.674847	2871	326	220
12s83	5847	342	178	0.520467836	6033	351	191	0.54416	6126	349	193
12t118	-828	372	28	0.075268817	300	362	82	0.226519	282	349	72
130	12	276	33	0.119565217	855	286	45	0.157343	846	287	43
132	486	627	117	0.186602871	1512	649	188	0.289676	1512	649	188
133	-1665	1449	120	0.082815735	1749	1528	358	0.234293	1794	1525	345
134	1353	405	28	0.069135802	2253	434	64	0.147465	2508	443	55
134s4	1881	387	58	0.149870801	2172	425	85	0.2	2418	428	80
134t4	816	387	68	0.175710594	1359	408	126	0.308824	1356	408	123
136s14	6495	438	287	0.655251142	6516	438	288	0.657534	6504	440	288
136s16	28320	462	243	0.525974026	28512	464	248	0.534483	28515	463	247
136s19	5724	426	241	0.5657277	5850	442	267	0.604072	5850	442	267
136s20	8196	438	200	0.456621005	8292	451	211	0.467849	8295	451	212

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree				
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM	
136s22	21021	444	115	0.259009009	22527	477	140	0.293501	22479	474	135	0.28481
136s8	4302	423	356	0.841607565	4302	423	356	0.841608	4302	423	356	0.841608
136t21	30306	462	161	0.348484848	31119	476	184	0.386555	31125	475	183	0.385263
136t25	20865	471	69	0.146496815	23727	501	98	0.195609	23367	507	91	0.179487
136t27	-456	489	16	0.032719836	936	548	65	0.118613	855	549	63	0.114754
137	11892	549	154	0.280510018	12771	574	210	0.365854	12723	573	205	0.357766
137s3	8457	543	401	0.738489871	8472	547	404	0.738574	8472	546	404	0.739927
137t4	11181	543	309	0.569060773	11217	546	314	0.575092	11220	545	313	0.574312
139	2712	357	28	0.078431373	4056	414	54	0.130435	3879	423	52	0.122931
139s4	1302	330	122	0.369696997	1560	348	152	0.436782	1545	351	152	0.433048
139s6	1254	357	51	0.142857143	1983	386	109	0.282383	1959	385	108	0.280519
139t6	2919	345	71	0.205797101	3465	369	89	0.241192	3579	378	90	0.238095
14	-480	258	20	0.07751938	99	242	43	0.177686	93	243	39	0.160494
140t11	84	567	76	0.134038801	1050	601	155	0.257903	1050	593	149	0.251265
140t7	810	513	95	0.185185185	1305	553	148	0.267631	1305	551	153	0.277677
144	2838	522	94	0.180076628	3747	561	170	0.30303	3747	561	170	0.30303

Continuación de la Tabla A.1 de la página anterior

Archivo	Original					RL-Permutation					RL-Tree		
	SP-Score	AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM	CS	
144s2	3162	522	253	0.48467433	3270	541	274	0.50647	3288	533	270	0.506567	
150	7125	969	567	0.585139319	7281	986	596	0.604462	7281	987	597	0.604863	
186	5091	1278	400	0.312989045	7491	1271	657	0.516916	7494	1271	658	0.517703	
19	-48	384	43	0.111979167	1014	387	125	0.322997	1017	389	122	0.313625	
197	14469	1308	454	0.347094801	18267	1329	592	0.445448	18102	1324	586	0.442598	
197t3	11100	1275	954	0.748235294	11796	1275	998	0.782745	11763	1274	994	0.78022	
22s18	5916	327	296	0.905198777	6108	324	310	0.95679	6108	324	310	0.95679	
22s30	4464	318	220	0.691823899	4584	321	228	0.71028	4584	321	229	0.713396	
22s33	3813	297	181	0.609427609	3948	299	190	0.635452	3954	300	191	0.636667	
22s34	2205	297	176	0.592592593	2241	301	184	0.611296	2241	301	184	0.611296	
22s36	9711	327	209	0.639143731	10080	331	223	0.673716	10116	333	225	0.675676	
22s41	18303	330	103	0.312121212	20097	349	129	0.369628	20208	350	133	0.38	
22s56	-603	324	20	0.061728395	297	324	79	0.243827	297	324	79	0.243827	
22t39	4257	297	93	0.313131313	4731	313	132	0.421725	4713	312	130	0.416667	
22t40	633	291	37	0.127147766	1167	326	67	0.205521	1158	324	68	0.209877	
22t42	2811	315	62	0.196825397	3450	346	96	0.277457	3363	342	95	0.277778	

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree			
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM
22t47	696	297	28	0.094276094	1299	346	56	0.16185	1497	332	57
22t49	7659	315	29	0.092063492	11043	376	56	0.148936	10914	372	54
22t55	-1089	363	10	0.027548209	1308	395	32	0.081013	1332	385	36
22t58	-3573	390	6	0.015384615	951	405	13	0.032099	861	412	19
22t59	-441	291	20	0.068728522	429	294	76	0.258503	429	294	76
24	-3075	543	23	0.042357274	8562	444	41	0.092342	8295	462	48
24t4	5700	387	275	0.710594315	5736	388	278	0.716495	5736	388	278
24t7	393	357	68	0.19047619	1044	339	99	0.292035	1044	339	99
24t8	-537	453	43	0.094922737	1476	401	65	0.162095	1584	417	69
256	-1662	1056	120	0.113636364	819	1124	276	0.245552	807	1076	263
263t3	6504	1227	541	0.440912795	6750	1256	581	0.46258	6750	1257	581
267	3240	1023	291	0.284457478	4611	1033	403	0.390126	4626	1035	404
294	2691	618	80	0.129449838	5091	659	117	0.177542	5484	650	113
294t3	2622	564	191	0.338652482	3072	571	236	0.41331	3072	572	237
307s4	11799	825	497	0.602424242	11913	835	516	0.617964	11913	835	516
34	933	273	20	0.073260073	2034	284	58	0.204225	1998	294	63

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree			
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM
341	8154	516	155	0.300387597	8616	541	177	0.327172	8652	551	178
341s4	2769	516	239	0.463178295	3015	529	261	0.493384	3024	527	261
34t3	1761	267	72	0.269662921	1989	283	86	0.303887	1884	284	84
35	-3492	1119	33	0.029490617	2028	1036	168	0.162162	1989	1060	192
354	1119	432	93	0.215277778	1749	454	161	0.354626	1749	454	161
355	228	630	81	0.128571429	1140	671	160	0.23845	1173	682	169
35t2	-342	933	89	0.095391211	1926	927	233	0.251348	1923	929	234
383	909	345	90	0.260869565	1143	368	122	0.331522	1143	368	123
39	1332	636	76	0.119496855	3666	631	180	0.285261	3666	631	180
39t2	2808	615	230	0.37398374	3501	620	304	0.490323	3513	620	306
4	-1941	414	10	0.024154589	1056	385	48	0.124675	792	383	40
41	2895	486	105	0.216049383	4323	478	199	0.416318	4323	478	199
414	1680	162	45	0.277777778	1992	165	63	0.381818	1959	167	62
415	513	135	42	0.311111111	600	147	55	0.37415	597	145	56
41t2	3741	387	298	0.77002584	3855	384	307	0.799479	3855	384	307
42	-5373	1044	12	0.011494253	345	993	165	0.166163	285	974	142

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree				
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM	
429s6	8928	174	153	0.879310345	8928	174	153	0.87931	8928	174	153	0.87931
429s9	9318	183	76	0.415300546	9648	194	95	0.489691	9648	194	95	0.489691
43	1527	1128	50	0.044326241	16236	1238	145	0.117124	16599	1222	139	0.113748
431	1878	195	88	0.451282051	2220	204	116	0.568627	2220	204	116	0.568627
437	6966	435	34	0.07816092	7752	477	51	0.106918	7641	496	51	0.102823
437s5	3327	420	91	0.216666667	3591	448	117	0.261161	3729	452	117	0.25885
437t7	6387	429	44	0.102564103	6981	471	62	0.131635	7185	482	68	0.141079
43s6	9267	753	255	0.338645418	10161	799	307	0.38423	10245	798	315	0.394737
43t8	16236	774	151	0.195090439	18501	870	202	0.232184	18978	854	200	0.234192
451	888	336	58	0.172619048	1452	343	75	0.218659	1443	355	80	0.225352
468	3783	468	168	0.358974359	4023	486	185	0.380658	3966	492	188	0.382114
471	1662	318	140	0.440251572	1746	328	155	0.472561	1752	325	154	0.473846
488	2592	342	130	0.380116959	3168	351	167	0.475783	3162	353	168	0.475921
490	5730	588	39	0.066326531	9585	626	57	0.091054	9498	630	56	0.088889
490s4	2355	486	189	0.388888889	2628	499	217	0.43487	2628	500	218	0.436
490s5	3522	492	157	0.319105691	4005	510	183	0.358824	3996	508	183	0.360236

Continuación de la Tabla A.1 de la página anterior

Archivo	Original				RL-Permutation				RL-Tree			
	SP-Score	AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM	CS
490t7	5892	522	112	0.214559387	6642	561	136	0.242424	6420	554	134	0.241877
490t8	210	543	72	0.132596685	1122	570	158	0.277193	1122	570	158	0.277193
491	5193	588	42	0.071428571	7461	657	67	0.101979	7977	651	82	0.12596
491s4	2556	528	211	0.399621212	2826	549	242	0.440801	2826	549	242	0.440801
491s6	4536	567	116	0.204585538	5529	601	161	0.267887	5352	605	152	0.25124
491t6	3648	588	46	0.078231293	5835	650	75	0.115385	6021	652	89	0.136503
498	1602	330	98	0.296969697	2091	339	125	0.368732	2091	340	126	0.370588
4t2	1005	264	93	0.352272727	1050	272	99	0.363971	1050	270	100	0.37037
4t3	441	318	42	0.132075472	1191	311	75	0.241158	1173	313	73	0.233227
512	1719	459	154	0.335511983	2019	476	186	0.390756	2019	475	185	0.389474
52	246	450	50	0.111111111	1338	452	131	0.289823	1338	452	131	0.289823
525	819	453	101	0.222958057	1539	444	144	0.324324	1539	445	145	0.325843
553	6600	402	72	0.179104478	7521	444	91	0.204955	7992	439	99	0.225513
553s6	4674	378	128	0.338624339	4971	392	146	0.372449	4974	385	145	0.376623
573	879	138	74	0.536231884	987	144	89	0.618056	987	144	89	0.618056
57s7	-4257	870	13	0.014942529	126	841	138	0.16409	117	826	129	0.156174

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree			
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM
57t4	522	624	125	0.200320513	1290	651	181	0.278034	1284	652	181
57t7	-1914	663	17	0.025641026	1356	721	102	0.14147	1347	701	93
581s10	1149	267	106	0.397003745	1386	272	132	0.485294	1386	272	132
581s11	4272	327	106	0.324159021	4374	338	117	0.346154	4371	339	115
581s12	2649	222	85	0.382882883	2832	229	92	0.401747	2823	226	91
581s14	1317	267	56	0.209737828	1752	276	87	0.315217	1740	276	89
581s5	1455	219	120	0.547945205	1485	222	127	0.572072	1491	224	128
581t13	3129	339	50	0.147492625	4200	334	67	0.200599	4032	341	59
581t14	4092	339	50	0.147492625	5265	335	60	0.179104	5292	331	64
581t17	-396	282	17	0.060283688	210	296	50	0.168919	159	294	43
581t18	-4443	360	9	0.025	876	345	14	0.04058	342	355	11
581t9	1863	342	50	0.14619883	2817	338	79	0.233728	2754	352	82
587	957	249	101	0.40562249	1317	250	126	0.504	1278	253	128
61	-351	831	79	0.095066185	4896	848	150	0.176887	4947	842	154
61s2	3816	531	315	0.593220339	3885	536	324	0.604478	3888	536	325
61t3	4977	534	227	0.425093633	5262	553	242	0.437613	5253	552	241

Continuación de la Tabla A.1 de la página anterior

Archivo	Original				RL-Permutation				RL-Tree			
	SP-Score	AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM	CS
62	21	267	35	0.131086142	672	267	82	0.307116	666	267	81	0.303371
620s7	15276	336	249	0.741071429	15393	338	253	0.748521	15393	338	253	0.748521
620t12	23004	351	105	0.299145299	24321	372	113	0.303763	24270	375	113	0.301333
641	780	312	75	0.240384615	1905	300	104	0.346667	1890	301	102	0.33887
655	573	264	66	0.25	798	274	85	0.310219	828	277	88	0.31769
658	-36	387	76	0.196382429	771	358	119	0.332402	771	358	119	0.332402
66	-3096	396	5	0.012626263	1320	380	39	0.102632	1575	404	26	0.064356
669	504	171	55	0.321637427	669	184	72	0.391304	675	186	74	0.397849
66s5	-1383	396	15	0.0378778788	2139	392	56	0.142857	2343	384	50	0.130208
66t4	-51	366	43	0.117486339	978	378	117	0.309524	867	378	117	0.309524
66t5	597	309	52	0.16828479	1308	311	116	0.37299	1308	311	116	0.37299
74	1560	1419	91	0.064129669	10053	1367	301	0.22019	9339	1369	311	0.227173
74t3	6807	1206	309	0.256218905	9063	1294	462	0.357032	9021	1291	458	0.354764
75	-345	249	20	0.080321285	363	250	49	0.196	363	251	46	0.183267
75t2	36	195	28	0.143589744	387	206	58	0.281553	408	198	54	0.272727
76	1407	294	114	0.387755102	1497	306	130	0.424837	1509	302	128	0.423841

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree			
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM
78s9	-1266	480	24	0.05	1983	479	56	0.11691	2256	487	57
78t10	-6297	528	13	0.024621212	4194	546	38	0.069597	4053	534	30
78t7	2445	468	185	0.395299145	2826	479	227	0.473904	2829	480	229
78t8	345	441	85	0.192743764	948	476	134	0.281513	912	459	121
78t9	612	483	36	0.074534161	4425	495	96	0.193939	4224	510	95
8	2829	552	109	0.197463768	4560	553	221	0.399638	4560	550	218
80	-2730	852	55	0.064553991	906	794	133	0.167506	768	782	123
82	-3822	558	7	0.012544803	3351	587	31	0.052811	3258	570	30
82s4	1134	402	101	0.251243781	1587	416	150	0.360577	1584	419	152
82s5	522	408	73	0.178921569	1263	421	132	0.313539	1260	420	131
82t5	2082	411	46	0.111922141	3498	452	76	0.168142	3387	450	85
82t6	1572	438	31	0.070776256	3711	482	63	0.130705	3657	483	58
82t7	996	444	24	0.054054054	4056	497	48	0.096579	3903	496	45
85	831	318	42	0.132075472	1887	326	70	0.214724	1710	338	75
85t3	1062	291	97	0.333333333	1182	303	113	0.372937	1182	303	113
86s7	10719	297	244	0.821548822	10839	297	251	0.845118	10839	297	251

Continuación de la Tabla A.1 de la página anterior

Archivo	SP-Score	Original			RL-Permutation			RL-Tree			
		AI	EM	CS	SP-Score	AI	EM	CS	SP-Score	AI	EM
86t10	15486	339	70	0.206489676	17784	361	123	0.34072	17775	364	122
86t9	15654	297	125	0.420875421	16404	311	147	0.472669	16458	314	145
8t2	3789	477	329	0.689727463	3954	479	338	0.705637	3954	480	340
9	5244	369	90	0.243902439	6927	346	177	0.511561	6924	343	176
90	-3780	1038	47	0.045279383	1155	1022	151	0.14775	1056	1018	143
92s2	2406	372	203	0.545698925	2709	375	224	0.597333	2709	375	224
92t3	2103	396	112	0.282828283	3120	420	170	0.404762	3156	415	165
92t4	2574	393	79	0.201017812	4203	409	134	0.327628	4206	414	128
93	771	252	23	0.091269841	1677	270	27	0.1	1599	275	26
93s5	498	195	38	0.194871795	696	207	47	0.227053	642	210	42
93t7	1707	198	24	0.121212121	1992	214	25	0.116822	1902	217	28
9t3	6372	327	321	0.981651376	6372	327	321	0.981651	6372	327	321

RL-Permutation				RL-Tree			
Archivo	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max
104s10	18	0.01090564	18	0.01090564	5	217.8	222
104s13	54	0.0259444	54	0.0259444	4	222	210
104s14	30	-0.00138972	-15	0.00553602	6	219	222
104s16	66	0.03611174	39	0.01659254	5	222	219
104s7	9	0.00314056	9	0.00314056	4	219	222
104t18	93	0.02687339	99	0.02955959	5	258	219
10s10	1389	0.02423425	2340	0.02309351	9	463.666667	258
10s3	261	0.05132426	255	0.04929174	4	483	426
10t8	306	0.03984844	315	0.04086261	5	483	480
10t9	465	0.08451417	468	0.08731451	4	433.5	480
113	1365	0.14766414	1290	0.1374766	4	397.5	414
115	954	0.12032828	975	0.11970859	4	423.75	309
116	3153	0.16530551	3111	0.1627497	5	505.2	396
118	126	0.0658764	156	0.07665386	4	195	423
120	186	0.0703691	186	0.04529402	4	266.25	249
121	1404	0.16407408	1434	0.16762982	4	512.25	492
123	2283	0.11042192	2256	0.11928713	5	374.4	357

Continuación de la Tabla A.2 de la página anterior

		RL-Permutation			RL-Tree					
Archivo	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min		
12s55	102	0.01009495	99	0.00720497	7	324	345	315		
12s58	0	0	0	0	4	332.25	333	330		
12s59	195	0.01920638	213	0.02225517	6	319.5	324	315		
12s81	27	0.01126638	27	0.01126638	4	321.75	324	321		
12s83	186	0.02369171	279	0.03254076	6	333	342	318		
12t118	1128	0.15125052	1110	0.13103491	4	294	309	267		
130	843	0.03777744	834	0.03026057	6	228.5	264	201		
132	1026	0.10307355	1026	0.10307355	4	547.5	615	477		
133	3414	0.15147746	3459	0.14341377	4	1257.75	1371	1209		
134	900	0.07832964	1155	0.0550177	7	360.428571	381	339		
134s4	291	0.0501292	537	0.03704509	6	359	381	339		
134t4	543	0.13311294	540	0.12575999	4	366	369	363		
136s14	21	0.00228311	9	-0.00070569	5	437.4	438	435		
136s16	192	0.00850873	195	0.0075033	10	450.6	462	441		
136s19	126	0.0383447	126	0.0383447	5	423.6	426	423		
136s20	96	0.01122822	99	0.01344551	6	437.5	438	435		

Continuación de la Tabla A.2 de la página anterior

Archivo	RL-Permutation			RL-Score			RL-Tree		
	Dif Spscore	Dif CS	Dif CScore	N	L promedio	L max	L min		
136s22	1506	0.03449204	1458	0.02580112	11	431.181818	438	423	
136s8	0	0	0	0	4	423	423	423	
136t21	813	0.03806977	819	0.03677831	11	449.181818	462	435	
136t25	2862	0.04911197	2502	0.03299036	12	432	441	423	
136t27	1392	0.0858933	1311	0.08203426	6	430.5	447	408	
137	879	0.08534364	831	0.07725612	7	537	543	519	
137s3	15	8.4169E-05	15	0.00143687	5	541.8	543	540	
137t4	36	0.0060308	39	0.00525115	6	540	543	531	
139	1344	0.05200341	1167	0.04450007	9	317	333	303	
139s4	258	0.06708464	243	0.06335146	4	313.5	315	312	
139s6	729	0.13952628	705	0.13766234	5	319.8	333	303	
139t6	546	0.03539531	660	0.03229814	7	313.285714	321	303	
14	579	0.10016657	573	0.08297445	4	190.5	231	162	
140t11	966	0.12386469	966	0.11722595	4	513.75	540	498	
140t7	495	0.08244592	495	0.09249177	4	490.5	513	474	
144	909	0.12295367	909	0.12295367	5	507.6	522	474	

Continuación de la Tabla A.2 de la página anterior

		RL-Permutation						RL-Tree			
Archivo	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min			
144s2	108	0.02179517	126	0.02189227	4	516	522	513			
150	156	0.01932316	156	0.0197239	4	955.5	966	942			
186	2400	0.20392677	2403	0.20471355	4	1085.25	1242	1029			
19	1062	0.21101825	1065	0.20164551	4	318	345	297			
197	3798	0.0983529	3633	0.09550339	6	1180.5	1260	1119			
197t3	696	0.0345098	663	0.03198449	4	1202.25	1260	1167			
22s18	192	0.05159135	192	0.05159135	5	316.2	321	312			
22s30	120	0.01845647	120	0.02157174	5	307.8	318	300			
22s33	135	0.0260239	141	0.02723906	5	281.4	297	267			
22s34	36	0.01870309	36	0.01870309	4	296.25	297	294			
22s36	369	0.03457228	405	0.03653194	7	316.285714	321	312			
22s41	1794	0.0575063	1905	0.06778789	12	312.75	321	300			
22s56	900	0.18209877	900	0.18209877	4	249.75	309	216			
22t39	474	0.10859393	456	0.10353535	6	280.5	297	267			
22t40	534	0.07837371	525	0.08272878	5	272.4	276	267			
22t42	639	0.08063125	552	0.08095238	6	299	309	294			

Continuación de la Tabla A.2 de la página anterior

Archivo	RL-Permutation			RL-Score			RL-Tree		
	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min	
22t47	603	0.06757362	801	0.07741065	6	274	282	267	
22t49	3384	0.05687268	3255	0.0530978	12	289.75	309	267	
22t55	2397	0.05346445	2421	0.06595828	8	282.75	312	267	
22t58	4524	0.01671415	4434	0.03073189	10	288.3	312	267	
22t59	870	0.18977488	870	0.18977488	4	240.75	273	216	
24	11637	0.04998507	11370	0.06153883	11	351.272727	387	294	
24t4	36	0.00590053	36	0.00590053	5	385.8	387	384	
24t7	651	0.10155921	651	0.10155921	4	300	315	294	
24t8	2013	0.06717203	2121	0.07054489	6	322.5	369	294	
256	2481	0.13191524	2469	0.13078743	4	838.5	1020	684	
263t3	246	0.02166682	246	0.02129882	4	1203	1212	1200	
267	1371	0.10566837	1386	0.10588069	4	948	981	924	
294	2400	0.04809189	2793	0.04439632	7	538.285714	549	534	
294t3	450	0.0746575	450	0.07568318	4	540	549	537	
307s4	114	0.01553983	114	0.01553983	5	824.4	825	822	
34	1101	0.13096528	1065	0.14102564	6	251	264	204	

Continuación de la Tabla A.2 de la página anterior

RL-Permutation		RL-Tree						
Archivo	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min
341	462	0.02678431	498	0.0226614	7	500.571429	513	492
341s4	246	0.03020545	255	0.03207787	4	505.5	513	498
34t3	228	0.034224	123	0.02611173	5	260.4	264	255
35	5520	0.13267155	5481	0.15164146	5	839.4	939	792
354	630	0.13934777	630	0.13934777	4	410.25	417	399
355	912	0.10987865	945	0.11922916	4	571.5	612	552
35t2	2268	0.15595722	2265	0.15649253	4	814.5	840	792
383	234	0.07065217	234	0.07336957	4	331.5	339	321
39	2334	0.16576463	2334	0.16576463	5	525	585	426
39t2	693	0.11633884	705	0.11956465	4	549.75	585	528
4	2997	0.10052074	2733	0.08028405	7	276.428571	327	246
41	1428	0.20026861	1428	0.20026861	5	397.8	468	372
414	312	0.1040404	279	0.09347971	6	154.5	162	135
415	87	0.06303855	84	0.07509579	4	126	126	126
41t2	114	0.02945333	114	0.02945333	4	380.25	384	372
42	5718	0.15466889	5658	0.1342963	5	724.8	873	573

Continuación de la Tabla A.2 de la página anterior

Archivo	RL-Permutation			RL-Score			RL-Tree		
	Dif Spscore	Dif CS	Dif CScore	N	L promedio	L max	L min		
429s6	0	0	0	8	169.875	174	153		
429s9	330	0.07439018	330	9	171.333333	183	153		
43	14709	0.07279815	15072	11	805.363636	1089	705		
431	342	0.1173454	342	5	190.2	192	189		
437	786	0.02875732	675	10	398.4	411	393		
437s5	264	0.04449405	402	6	399.5	411	393		
437t7	594	0.02907072	798	9	399	411	393		
43s6	894	0.04558487	978	6	735.5	747	705		
43t8	2265	0.03709347	2742	9	743.333333	762	705		
451	564	0.04603984	555	5	303.6	315	291		
468	240	0.02168408	183	5	456.6	462	450		
471	84	0.0323094	90	4	312	312	312		
488	576	0.09566652	570	5	326.4	342	303		
490	3855	0.02472778	3768	11	472.636364	525	441		
490s4	273	0.04598085	273	4	468	477	459		
490s5	483	0.03971784	474	5	469.2	477	459		

Continuación de la Tabla A.2 de la página anterior

RL-Permutation		RL-Tree						
Archivo	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min
490t7	750	0.02786486	528	0.02731787	7	472.285714	483	459
490t8	912	0.1445963	912	0.1445963	4	473.25	525	441
491	2268	0.03055012	2784	0.05453149	9	519.666667	543	510
491s4	270	0.04118025	270	0.04118025	4	513.75	525	510
491s6	993	0.06330132	816	0.04665413	6	518.5	543	510
491t6	2187	0.03715332	2373	0.05827177	8	520.5	543	510
498	489	0.07176187	489	0.07361854	5	281.4	303	255
4t2	45	0.01169786	45	0.01809764	4	252.75	258	246
4t3	750	0.10908208	732	0.10115137	5	261	294	246
512	300	0.05524432	300	0.05396117	4	447.75	456	432
52	1092	0.1787119	1092	0.1787119	4	401.25	408	390
525	720	0.10136627	720	0.10288464	4	386.25	420	366
553	921	0.02585048	1392	0.04640805	9	361	372	345
553s6	297	0.03382464	300	0.03799904	6	361.5	372	354
573	108	0.08182367	108	0.08182367	4	136.5	138	135
57s7	4383	0.14914784	4374	0.14123181	5	623.4	780	513

Continuación de la Tabla A.2 de la página anterior

Archivo	RL-Permutation			RL-Score			RL-Tree		
	Dif Spscore	Dif CS	Dif CScore	N	L promedio	L max	L min		
57t4	768	0.07771328	762	0.07728685	4	546.75	615	507	
57t7	3270	0.11582915	3261	0.10702659	6	546	615	507	
581s10	237	0.08829037	237	0.08829037	4	234	264	210	
581s11	102	0.02199482	99	0.01507402	6	324.5	327	321	
581s12	183	0.01886384	174	0.01977198	6	211.5	219	201	
581s14	435	0.10547956	423	0.11272594	5	230.4	264	210	
581s5	30	0.02412687	36	0.02348337	4	207.75	213	201	
581t13	1071	0.05310618	903	0.0255279	9	227	321	201	
581t14	1173	0.03161185	1200	0.04586085	10	226.2	321	201	
581t17	606	0.10863523	555	0.08597482	5	225	264	198	
581t18	5319	0.01557971	4785	0.00598592	11	270.545455	327	198	
581t9	954	0.08752898	891	0.08675572	6	308.5	327	225	
587	360	0.09837751	321	0.10030636	4	222.75	243	213	
61	5247	0.08182061	5298	0.08783168	7	561.428571	747	474	
61s2	69	0.01125727	72	0.01312294	4	498	525	474	
61t3	285	0.01251939	276	0.01150057	5	498	525	474	

Continuación de la Tabla A.2 de la página anterior

RL-Permutation		RL-Tree						
Archivo	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min
62	651	0.17602996	645	0.17228464	4	227.25	267	189
620s7	117	0.00744928	117	0.00744928	8	325.5	336	324
620t12	1317	0.00461814	1266	0.00218803	12	323.25	336	309
641	1125	0.10628205	1110	0.09848582	5	264.6	282	255
655	225	0.06021898	255	0.06768953	4	244.5	255	237
658	807	0.13601981	807	0.13601981	4	266.25	354	237
66	4416	0.09000532	4671	0.05173017	8	302.25	330	207
669	165	0.06966692	171	0.07621204	4	164.25	168	153
66s5	3522	0.10497835	3726	0.09232955	7	315.857143	330	294
66t4	1029	0.19203747	918	0.19203747	4	321	330	294
66t5	711	0.20470556	711	0.20470556	4	283.5	309	207
74	8493	0.15606053	7779	0.16304345	6	1163	1170	1143
74t3	2256	0.10081355	2214	0.09854484	5	1167	1170	1164
75	708	0.11567871	708	0.10294565	5	187.8	243	165
75t2	351	0.13796365	372	0.12913753	4	174	192	165
76	90	0.0370815	102	0.03608596	4	291.75	294	285

Continuación de la Tabla A.2 de la página anterior

Archivo	RL-Permutation			RL-Score			RL-Tree		
	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min	
78s9	3249	0.06691023	3522	0.06704312	7	388.714286	429	372	
78t10	10491	0.04497586	10350	0.03155856	11	409.363636	459	372	
78t7	381	0.07860482	384	0.08178419	4	445.5	459	435	
78t8	603	0.08876884	567	0.07087279	4	396.75	429	372	
78t9	3813	0.11940523	3612	0.11174035	7	416.571429	459	372	
8	1731	0.20217457	1731	0.19889987	5	466.2	516	432	
80	3636	0.10295231	3498	0.09273501	5	622.2	705	441	
82	7173	0.0402661	7080	0.04008678	11	384	456	354	
82s4	453	0.10933314	450	0.11152472	4	365.25	390	354	
82s5	741	0.13461762	738	0.13298319	4	369	387	357	
82t5	1416	0.05621945	1305	0.07696675	7	364.285714	390	354	
82t6	2139	0.05992914	2085	0.04930656	8	367.125	390	354	
82t7	3060	0.04252542	2907	0.03667175	9	369	390	354	
85	1056	0.08264845	879	0.08981802	6	281	291	264	
85t3	120	0.03960396	120	0.03960396	4	282	291	264	
86s7	120	0.02356902	120	0.02356902	7	288.428571	297	264	

Continuación de la Tabla A.2 de la página anterior

		RL-Permutation			RL-Tree			
Archivo	Dif Spscore	Dif CS	Dif Spscore	Dif CS	N	L promedio	L max	L min
86t10	2298	0.13423055	2289	0.12867516	11	292.909091	339	264
86t9	750	0.05179339	804	0.04090802	10	288.3	297	264
8t2	165	0.01590928	165	0.01860587	4	453.75	474	432
9	1683	0.26765825	1680	0.26921709	6	316.5	327	288
90	4935	0.10247013	4836	0.09519213	5	837	867	795
92s2	303	0.05163441	303	0.05163441	4	348	372	333
92t3	1017	0.12193362	1053	0.11476208	5	360.6	375	339
92t4	1629	0.12661055	1632	0.10816093	6	356	375	333
93	906	0.00873016	828	0.00327561	10	180.9	249	153
93s5	198	0.03218135	144	0.00512821	5	172.2	189	153
93t7	285	-0.00438969	195	0.00782014	9	173.333333	189	153
9t3	0	0	0	0	5	322.2	327	321

Apéndice B

Detalles de la implementación

La implementación de este trabajo de tesis se realizó en la plataforma Colab (*Google Colab, 2020*). Se utilizó el lenguaje de programación `python`, versión 3.6.9, así como las siguientes librerías:

1. `numpy` versión 1.18.5.
2. `tensorflow` versión 2.3.0.
3. `cppyy` versión 1.8.0.

`Numpy` y `tensorflow` son dos librerías ampliamente utilizadas en el ámbito del cómputo estadístico para el manejo de matrices numéricas y de redes neuronales. `Cppyy` *Documentación de la librería cppyy* (2020) es una librería que permite implementar funciones en Python que serán ejecutadas bajo el lenguaje de programación C++, dichas funciones pueden ser utilizadas en el mismo ambiente donde se ejecuta las instrucciones en `python`.

El algoritmo Needleman-Wunsch fue implementado utilizando `cppyy`. Este algoritmo es utilizado alrededor de N veces por episodio para obtener un alineamiento progresivo con los enfoques propuestos; por tanto una implementación cuyo tiempo de ejecución fuese eficiente era fundamental. La versión en C++ del algoritmo de Needleman-Wunsch corre aproximadamente 100 veces más rápido que su contraparte

en python.

En cuanto a las redes neuronales utilizadas para aproximar la función Q del valor estado-acción, se utilizó una red con entradas múltiples con la finalidad de obtener representaciones útiles, tanto del estado como de la acción correspondiente, para concatenarlas y, desde la nueva representación del estado-acción, obtener la aproximación del Q valor correspondiente. También se realizaron experimentos utilizando como entrada de la red solo la representación del estado y como salida un vector con los Q valores de todas las acciones. Esto introdujo algunas complicaciones, pues no todos los pares estado-acción son compatibles. Goodfellow, Bengio, y Courville (2016) ofrece una amplia cobertura sobre la estructuración de redes neuronales y algoritmos de optimización para ajustarlas. Dado el carácter secuencial de las representaciones de los estados en los tres enfoques propuestos en esta tesis, se decidió usar una red LSTM (Long short-term memory) para obtener una representación útil de los estados.

A pesar de que en la plataforma de Colab se pueden utilizar aceleradores de hardware GPU o TPU, se decidió no utilizarlos pues no se observaron ahorros considerables en los tiempos de ejecución y se pierde reproducibilidad en los resultados. Esto último debido a que controlar el orden en el que un GPU o TPU realiza sus operaciones es sumamente complicado.

Apéndice C

Código

En la siguiente liga <https://github.com/AngelDominguezLozano/TesisMCE-CIMAT-AngelDominguez> se encuentran los resultados mencionados en el Capítulo 4, así como una carpeta con la implementación de los métodos. También se adjunta aquí el código de los métodos propuestos en este trabajo.

```
import time
import numpy as np
import progressbar

from tensorflow.keras.models import load_model
from tensorflow.keras import layers
from numpy.random import seed
from tensorflow.random import set_seed
from tensorflow import keras

import cppyy

cppyy.cppdef("""
using namespace std;
vector<double> NW(vector<double> S, vector<double> p1, vector<double> p2) {
    int i,j,k,r;
    int aux=S.size();
    int Q;
    if(aux==25) Q=5;
    else Q=21;
    int l1= p1.size()/Q;
    int l2=p2.size()/Q;
    vector<double> ans( (l1+1)*(l2+1) ), prev( (l1+1)*(l2+1) );
    ans[0]=0;
    prev[0]=-1;
    for(j=1;j<=l2;j++) {
        ans[(l2+1)*(0)+(j)]=ans[(l2+1)*(0)+(j-1)];
        ans[(l2+1)*(0)+(j)]+=p1[j];
        ans[(l2+1)*(0)+(j)]*=p2[j];
    }
    return ans;
}
""")
```

```

prev[(12+1)*(0)+(j)]=0;
for(k=0;k<Q;k++){
    ans[(12+1)*(0)+(j)]+=S[(Q)*(k)+(Q-1)]*p2[(Q)*(j-1)+(k)];
}
}

for(i=1;i<=l1;i++) {
    ans[(12+1)*(i)+(0)]=ans[(12+1)*(i-1)+(0)];
    prev[(12+1)*(i)+(0)]=2;
    for(k=0;k<Q;k++){
        ans[(12+1)*(i)+(0)]+=S[(Q)*(k)+(Q-1)]*p1[(Q)*(i-1)+(k)];
    }
}

double aux1,aux2;
for(i=1;i<=l1;i++) {
    for(j=1;j<=l2;j++) {
        ans[(12+1)*(i)+(j)]=ans[(12+1)*(i-1)+(j-1)];
        prev[(12+1)*(i)+(j)]=1;
        for(k=0;k<Q;k++) {
            for(r=0;r<Q;r++) {
                ans[(12+1)*(i)+(j)]+=p1[(Q)*(i-1)+(k)]*S[(Q)*(k)+(r)]*p2[(Q)*(j-1)+(r)];
            }
        }
        aux1=ans[(12+1)*(i)+(j-1)];
        for(k=0;k<Q;k++) {
            aux1+=S[(Q)*(k)+(Q-1)]*p2[(Q)*(j-1)+(k)];
        }
        aux2=ans[(12+1)*(i-1)+(j)];
        for(k=0;k<Q;k++) {
            aux2+=S[(Q)*(k)+(Q-1)]*p1[(Q)*(i-1)+(k)];
        }
        if(aux1>ans[(12+1)*(i)+(j)]) {
            ans[(12+1)*(i)+(j)]=aux1;
            prev[(12+1)*(i)+(j)]=0;
        }
        if(aux2>ans[(12+1)*(i)+(j)]) {
            ans[(12+1)*(i)+(j)]=aux2;
            prev[(12+1)*(i)+(j)]=2;
        }
    }
}

vector<double> cam;
i=l1;
j=l2;
while( i!=0 || j!=0) {
    cam.push_back(prev[(12+1)*(i)+(j)]);
}

```

```

        if(prev[(12+1)*(i)+(j)]==1) {
            i--;
            j--;
        }
        else {
            if(prev[(12+1)*(i)+(j)]==0) {
                j--;
            }
            else {
                i--;
            }
        }
    }
    return cam;
}
"""
)

class AlineadorRL:
    def __init__(self,sequenceType='DNA',matchScore=2, unMatchScore=-1,
                 matchGapScore=-2,RLmethod='Tree'):
        self.sequenceType = sequenceType
        self.matchScore = matchScore
        self.unMatchScore = unMatchScore
        self.matchGapScore=matchGapScore
        self.RLmethod = RLmethod
        if self.sequenceType == 'DNA':
            self.Q = ['G','A','T','C','-']
        if self.sequenceType == 'RNA':
            self.Q = ['G','A','U','C','-']
        if self.sequenceType == 'AMINOACID':
            self.Q = ['A','R','N','D','C','Q','E','G','H','I','L','K','M','F','P',
                      'S','T','W','Y','V','-']
        self.mapQ ={}
        for i in range(len(self.Q)): self.mapQ[self.Q[i]]=i
        Qlen = len(self.Q)-1
        S= np.zeros( (Qlen+1,Qlen+1) )
        for i in range(Qlen+1):
            for j in range(Qlen+1):
                if (i== Qlen and j != Qlen ) or (i!= Qlen and j == Qlen) :
                    S[i,j]= self.matchGapScore
                    continue
                if i == j and i == Qlen :
                    S[i,j] = 0
                    continue
                if i == j :
                    S[i,j] = self.matchScore

```

```

    else :
        S[i,j] = self.unMatchScore
    self.S = S

def scoreProfileVectors(self, v1,v2):
    ans = np.matmul(v1,self.S)
    ans = np.matmul(ans,v2)
    return ans

def scoreProfileVectorGap(self,v):
    ans = self.S[len(self.S)-1].dot(v)
    return ans

def blockToProfile(self,B):
    cant = len(B)
    per = np.zeros( (len(self.Q), len(B[0]) ) )
    for i in range(len(B)):
        for j in range(len(B[i])):
            per[ self.mapQ[B[i][j]] ,j]+=1
    per = per / cant
    return per.T

def SPscore(self,B,r=-1):
    ans = 0
    for k in range(len(B[0])):
        for i in range(len(B)-1):
            for j in range(i+1,len(B)):
                if r== -1:
                    ans+=self.S[self.mapQ[B[i][k]], self.mapQ[B[j][k]]]
                else:
                    if i<= len(B)-r-1 and j >= len(B)-r:
                        ans+=self.S[self.mapQ[B[i][k]], self.mapQ[B[j][k]]]
    return ans

def metricas(self,B):
    EM=0
    for i in range(len(B[0])):
        localeM=1
        if len(B) > 1:
            for j in range(1,len(B)):
                if B[j][i]!=B[0][i]: localeM=0
            EM+=localeM
    AL = len(B[0])
    CS = EM / AL
    return self.SPscore(B), AL , EM, CS

```

```

def SPvector(self,v):
    ans = 0
    for i in range(len(v)-1):
        for j in range(i+1,len(v)):
            ans+=self.S[self.mapQ[v[i]], self.mapQ[v[j]]]
    return ans

def alignProfileNW(self,B1,B2):
    # S matriz de score Q + 1 x Q + 1
    # B1 B2 bloques previamente alineados
    p1=self.blockToProfile(B1)
    p2=self.blockToProfile(B2)
    l1 = p1.shape[0]
    l2 = p2.shape[0]
    gap = np.zeros_like(p1[0])
    gap[len(self.Q)-1]=1
    # NW para guardar los valores de la DP
    NW = np.zeros( (l1+1,l2+1) )
    # NWprev para guardar la opcion que llevo a un maximo.
    # 0 'izquierda' , 1 'diagonal', 2 'arriba'
    NWprev = np.zeros_like(NW)
    ## Casos Base, prejifos vacios
    NW[0,0]= 0
    NWprev[0,0] = -1
    ## Casos Base, Prefijo no vacio contra vacio. Rellenar con gaps
    for i in range(1,l1+1):
        NW[i,0]=NW[i-1,0]+self.scoreProfileVectorGap(p1[i-1])
        NWprev[i,0]=2
    for j in range(1,l2+1):
        NW[0,j]=NW[0,j-1]+self.scoreProfileVectorGap(p2[j-1])
        NWprev[0,j] = 1
    ## Caso recursivo
    for i in range(1,l1+1):
        for j in range(1,l2+1):
            NW[i,j] = NW[i-1,j-1]+self.scoreProfileVectors(p1[i-1],p2[j-1])
            NWprev[i,j] = 1
            aux2 = NW[i-1,j] + self.scoreProfileVectorGap(p1[i-1])
            aux0 = NW[i,j-1] + self.scoreProfileVectorGap(p2[j-1])
            if aux2 > NW[i,j]:
                NW[i,j]= aux2
                NWprev[i,j] = 2
            if aux0 > NW[i,j]:
                NW[i,j]= aux0
                NWprev[i,j] = 0

```

```

## Reconstrucción del camino

fil=11
col = 12
camino = []
while NWprev[fil,col] != -1:
    camino.append(NWprev[fil,col])
    if NWprev[fil,col] == 0:
        col-=1
    else:
        if NWprev[fil,col] == 1:
            fil-=1
            col-=1
        else:
            fil-=1
    camino = np.flip(np.array(camino))
rows = len(B1)+len(B2)
B=[]
for i in range(rows):
    B.append("")
i=0
j=0
for mov in camino:
    for x in range(len(B1)):
        if mov == 2 or mov == 1:
            B[x]+=B1[x][i]
        else:
            B[x]+=_-
    for x in range(len(B2)):
        if mov == 0 or mov == 1:
            B[len(B1)+x]+=B2[x][j]
        else:
            B[len(B1)+x]+=_-
    if mov == 2 or mov == 1:
        i+=1
    if mov == 0 or mov == 1:
        j+=1
return B

def alignProfileNWcpp(self,B1,B2):
    # S matriz de score Q + 1 x Q + 1
    # B1 B2 bloques previamente alineados
    p1=self.blockToProfile(B1)
    p2=self.blockToProfile(B2)
    camcpp = cppyy.gbl.NW(self.S.flatten(),p1.flatten(),p2.flatten())
    camino=np.flip(np.array(list(camcpp),dtype=np.int))

```

```

rows = len(B1)+len(B2)
B=[]
for i in range(rows):
    B.append("")
i=0
j=0
for mov in camino:
    for x in range(len(B1)):
        if mov == 2 or mov == 1:
            B[x]+=B1[x][i]
        else:
            B[x]+='_-'
    for x in range(len(B2)):
        if mov == 0 or mov == 1:
            B[len(B1)+x]+=B2[x][j]
        else:
            B[len(B1)+x]+='_-'
    if mov == 2 or mov == 1:
        i+=1
    if mov == 0 or mov == 1:
        j+=1
return B

def create_modelPermutation(self,cant_sec):
    # Inputs
    inputAction = keras.Input(shape=(cant_sec,),name = 'Action')
    inputState = keras.Input(shape = (cant_sec,), name = 'State')
    # Proceso Action
    outAction = layers.Dense(16,activation="relu")(inputAction)
    # Proceso State
    emb1= layers.Embedding(input_dim=cant_sec+1, output_dim=5) (inputState)
    outState = layers.LSTM(16) (emb1)
    # Combinar
    comb = layers.concatenate([outAction,outState])
    combiD1= layers.Dense(16,activation="relu") (comb)
    outQ = layers.Dense(1) (combiD1)
    Q = keras.Model(inputs = [inputAction,inputState], outputs = [outQ])
    Q.compile(optimizer='adam',loss = 'mean_squared_error')
    return Q

def getQvalsPermutation(self,Qfun,actualState,validAction,maskAction) :
    qState = np.array([ actualState for x in validAction])
    qAction = np.array([maskAction[x] for x in validAction])
    qvals = np.array([x[0] for x in Qfun({'State':qState,'Action':qAction})])
    return qvals

def buildAlignmentPermutation(self,sec,Qfun,maskAction):

```

```

N = len(sec)
actualState = np.zeros((N))
pos = 0
while 0 in actualState:
    validAction = np.array(list(filter( lambda y: y+1 not in actualState ,
                                         [x for x in range(len(maskAction))])))
    qvals = self.getQvalsPermutation(Qfun,actualState,validAction,maskAction)
    a = int(np.argmax(qvals))
    a = validAction[a]
    actualState[pos] = a+1
    if pos == 0:
        B = [ sec[a] ]
    else :
        B = self.alignProfileNWcpp(B,[sec[a]])
    pos = pos + 1
return B

def alignPermutation(self,sec,sizeExp=100, sizeMiniBatch = 100, M= 20,
                     epsilon = 0.2, gamma = 0.1,alpha=0.1,patience=15,
                     check_convergence=False,
                     epochas=100, returnbestCS = False):
    seed(42)
    set_seed(42)
    iniTiempo=time.time()
    # Precalculos del enfoque
    N = len(sec)
    Qfun = self.create_modelPermutation(N)
    maskAction = np.array([ [ x if x==i else 0 for x in range(N)] for i in range(N)])
    metricasGlobal = []
    firstMetrica = True
    # Precalculos convergencia
    lastCheck = -1
    bestScore = 0
    bestCS = 0
    conv=False
    trained = False
    # Inicializar repositorio experiencias
    currentSizeExp=0
    nextExpIndex=0
    experience = np.zeros((sizeExp,N+2)) # N state, 1 action, 1 reward
    ep = 0
    # Inicia Q - Learning
    bar = progressbar.ProgressBar(maxval = M,widgets=[progressbar.Bar('=', '[' , ']'),
                                                       ' ', progressbar.Percentage()])
    bar.start()
    while ep<M and conv == False:

```

```

# Inicia Episodio

# Precalculos Episodio

actualState = np.zeros((N)) # estado actual, prefijo de permutacion del 1 al N.
                            # Se rellena el sufijo con ceros

pos = 0 # Proximo indice a considerar

while 0 in actualState:

    # Inicia Transicion

    # Inicio criterio de exploracion

    validAction = np.array(list(filter( lambda y: y+1 not in actualState ,
                                         [x for x in range(len(maskAction))])))

    criterio = np.random.uniform(0,1)

    if criterio <= epsilon:

        a = np.random.randint(0,len(validAction))

    else:

        qvals = self.getQvalsPermutation(Qfun,actualState,validAction,maskAction)

        a = int(np.argmax(qvals))

    a = validAction[a] # entre 0 y len(maskAction)

    # Fin criterio de exploracion

    # Inicia ejecucion de accion

    exp = actualState

    exp= np.append(exp,a)

    actualState[pos] = a+1

    if pos == 0:

        B = [ sec[a] ]

        exp =np.append(exp,0)

    else :

        B = self.alignProfileNWcpp(B,[sec[a]])

        exp = np.append(exp,self.SPscore(B,1))

    experience[nextExpIndex] = exp

    nextExpIndex = nextExpIndex + 1

    if nextExpIndex == sizeExp: nextExpIndex = 0

    currentSizeExp = currentSizeExp + 1

    pos = pos + 1

    # Fin ejecucion de accion

    # Inicia experience replay

    if currentSizeExp >= sizeMiniBatch:

        indExp = np.random.randint(0,min(currentSizeExp,sizeExp),sizeMiniBatch)

        qState = []

        qAction = []

        qTarget = []

        # Inicia Obtencion de bigQvals

        tamBatch = np.zeros((len(indExp)),dtype = np.int32)

        qBatchState = []

        qBatchAction = []

        for i,j in zip(indExp,range(len(indExp))):

            qBatchState.append(actualState[i])

```

```

st = np.array(experience[i][0:N])
act = int(experience[i][N])
stD = np.array(st)
w = 0
while stD[w] != 0:
    w = w+1
stD[w]=act+1
if 0 not in stD:
    qBatchState.append(st)
    qBatchAction.append(maskAction[act])
    tamBatch[j] = 0
else:
    validActionD = np.array(list(filter( lambda y: y+1 not in stD ,
[x for x in range(len(maskAction))])))
    qBatchState.append(st)
    qBatchAction.append(maskAction[act])
    for w in validActionD:
        qBatchState.append(stD)
        qBatchAction.append(maskAction[w])
    tamBatch[j] = len(validActionD)
qBatchState = np.array(qBatchState)
qBatchAction = np.array(qBatchAction)
bigQvals = np.array([x[0] for x in Qfun({'State':qBatchState,
                                            'Action':qBatchAction})])
iniBigQvals=0
# Fin obtencion de bigQvals
for i,j in zip(indExp,range(len(indExp))):
    st = np.array(experience[i][0:N])
    act = int(experience[i][N])
    rew = experience[i][N+1]
    qval = bigQvals[iniBigQvals]
    iniBigQvals= iniBigQvals+1
    stD = np.array(st)
    w = 0
    while stD[w] != 0:
        w = w+1
    stD[w]=act+1
    if 0 not in stD: # stD estado Final
        targ = rew
    else:
        qvalsD = bigQvals[iniBigQvals:(iniBigQvals+tamBatch[j])]
        iniBigQvals = iniBigQvals + tamBatch[j]
        targ = rew + gamma*np.max(qvalsD)
    targ = alpha*targ + (1-alpha)*qval
    qState.append(st)

```

```

qAction.append(maskAction[act])
qTarget.append(targ)
# Inicio fit
qState=np.array(qState)
qAction =np.array(qAction)
qTarget=np.array(qTarget)
Qfun.fit({'State':qState,'Action':qAction},y=qTarget, verbose=0,
          epochs = epochas)
trained=True
# Fin fit
# Termina experiencia replay
# Termina Transicion
ep = ep + 1
B = self.buildAlignmentPermutation(sec,Qfun,maskAction)
metricasEpisodio = self.metricas(B)
if trained == True :
    metricasGlobal.append(metricasEpisodio)
else:
    if firstMetrica == True:
        metricasGlobal.append(metricasEpisodio)
        firstMetrica = False
    if lastCheck == -1:
        lastCheck = ep
        bestScore = metricasEpisodio[0]
        bestCS = metricasEpisodio[3]
        Qfun.save("auxiliar_model.h5")
# Termina Episodio
# Inicio Revisar convergencia
if check_convergence==True and trained==True:
    if lastCheck == -1:
        lastCheck = ep
        bestScore = metricasEpisodio[0]
        bestCS = metricasEpisodio[3]
        Qfun.save("auxiliar_model.h5")
    else :
        localScore = metricasEpisodio[0]
        localCS = metricasEpisodio[3]
        if returnbestCS == False:
            if localScore< bestScore or ( localScore == bestScore and localCS <= bestCS) :
                if ep - lastCheck > patience:
                    conv= True
            else :
                bestScore = localScore
                bestCS = localCS
                lastCheck = ep

```

```

        Qfun.save("auxiliar_model.h5")

    else:
        if localCS < bestCS or ( localCS == bestCS and localScore <= bestScore) :
            if ep - lastCheck > patience:
                conv= True
            else :
                bestScore = localScore
                bestCS = localCS
                lastCheck = ep
            Qfun.save("auxiliar_model.h5")

    # Fin Revisar convergencia
    bar.update(ep)

# Termina Q - Learning
if check_convergence == True:
    Qfun = load_model("auxiliar_model.h5")
    B = self.buildAlignmentPermutation(sec,Qfun,maskAction)
    print( self.metricas(B))
    print("Tiempo ", time.time()-iniTiempo, " segundos")
    bar.finish()
    return B , Qfun , np.array(metricasGlobal)

def create_modelTree(self,cant_sec):
    # Inputs
    inputAction = keras.Input(shape=(cant_sec,),name = 'Action')
    inputState = keras.Input(shape = (cant_sec,), name = 'State')
    # Proceso Action
    outAction = layers.Dense(16,activation="relu")(inputAction)
    # Proceso State
    emb1= layers.Embedding(input_dim=cant_sec, output_dim=5) (inputState)
    outState = layers.LSTM(16) (emb1)
    # Combinar
    comb = layers.concatenate([outAction,outState])
    combiD1= layers.Dense(16,activation="relu") (comb)
    outQ = layers.Dense(1) (combiD1)
    Q = keras.Model(inputs = [inputAction,inputState], outputs = [outQ])
    Q.compile(optimizer='adam',loss = 'mean_squared_error')
    return Q

def getQvalsTree(self,Qfun,actualState,validAction,maskAction) :
    qState = np.array([ actualState for x in validAction])
    qAction = np.array([ np.array([ 0 if i != maskAction[x,0] and i!=maskAction[x,1]
                                    else 1 for i in range(len(actualState))])
                        for x in validAction])
    qvals = np.array([x[0] for x in Qfun({'State':qState,'Action':qAction})])
    return qvals

def buildAlignmentTree(self,sec,Qfun,maskAction):

```

```

N = len(sec)
actualState = np.array([x for x in range(N)],dtype = np.int32)
meta = np.array([0 for x in range(len(sec))],dtype=np.int32)
perfiles = [ [x] for x in sec ]
while sum(meta!=actualState)!=0:
    validAction = np.array(list(filter( lambda y: maskAction[y,0] in actualState
                                         and maskAction[y,1] in actualState,
                                         [x for x in range(len(maskAction))])))
    qvals = self.getQvalsTree(Qfun,actualState,validAction,maskAction)
    a = int(np.argmax(qvals))
    a = validAction[a]
    perfiles[maskAction[a,0]] = self.alignProfileNWcpp(perfiles[maskAction[a,0]],
                                                       perfiles[maskAction[a,1]])
    actualState[actualState == maskAction[a,1] ] = maskAction[a,0]
return perfiles[0]

def alignTree(self,sec,sizeExp=100, sizeMiniBatch = 100, M= 20, epsilon = 0.2,
             gamma = 0.1,alpha=0.1,patience=15,check_convergence=False,
             epochas=100,returnbestCS = False):
    seed(42)
    set_seed(42)
    iniTiempo=time.time()
    # Precalculos del enfoque
    N = len(sec)
    numEstados = int((N*(N-1)) /2)
    Qfun = self.create_modelTree(N)
    maskAction= np.zeros((numEstados,2),dtype=np.int32)
    i=0
    for x in range(N):
        for y in range(N):
            if x<y:
                maskAction[i,0]=x
                maskAction[i,1]=y
            i+=1
    meta = np.array([0 for x in range(len(sec))],dtype=np.int32)
    metricasGlobal = []
    firstMetrica = True
    # Precalculos convergencia
    lastCheck = -1
    bestScore = 0
    conv=False
    bestCS = 0
    trained = False
    # Inicializar repositorio experiencias
    currentSizeExp=0
    nextExpIndex=0

```

```

experience = np.zeros((sizeExp,N+2)) # N state, 1 action, 1 reward
ep = 0
# Inicia Q - Learning
bar = progressbar.ProgressBar(maxval = M,widgets=[progressbar.Bar('=', '[' , ']'),
                                                    ' ', progressbar.Percentage()])
bar.start()
while ep<M and conv == False:
    # Inicia Episodio
    # Precalculos Episodio
    actualState = np.array([x for x in range(N)],dtype = np.int32)
    perfiles = [ [x] for x in sec ]
    while sum(meta!=actualState)!=0:
        # Inicia Transicion
        # Inicio criterio de exploracion
        validAction = np.array(list(filter( lambda y: maskAction[y,0] in actualState
                                            and maskAction[y,1] in actualState,
                                            [x for x in range(len(maskAction))])))
        criterio = np.random.uniform(0,1)
        if criterio <= epsilon:
            a = np.random.randint(0,len(validAction))
        else:
            qvals = self.getQvalsTree(Qfun,actualState,validAction,maskAction)
            a = int(np.argmax(qvals))
            a = validAction[a]
        # Fin criterio de exploracion
        # Inicia ejecucion de accion
        perfiles[maskAction[a,0]] = self.alignProfileNWcpp(perfiles[maskAction[a,0]],
                                                          perfiles[maskAction[a,1]])
        exp = actualState
        exp= np.append(exp,a)
        actualState[actualState == maskAction[a,1] ] = maskAction[a,0]
        exp = np.append(exp,self.SPscore(perfiles[maskAction[a,0]],
                                         len(perfiles[maskAction[a,1]])) )
        experience[nextExpIndex] = exp
        nextExpIndex = nextExpIndex + 1
        if nextExpIndex == sizeExp: nextExpIndex = 0
        currentSizeExp = currentSizeExp + 1
        # Fin ejecucion de accion
        # Inicia experience replay
        if currentSizeExp >= sizeMiniBatch:
            indExp = np.random.randint(0,min(currentSizeExp,sizeExp),sizeMiniBatch)
            qState = []
            qAction = []
            qTarget = []
            # Inicia Obtencion de bigQvals

```

```

tamBatch = np.zeros((len(indExp)), dtype = np.int32)
qBatchState = []
qBatchAction = []
for i,j in zip(indExp,range(len(indExp))):
    st = np.array(experience[i][0:N])
    act = int(experience[i][N])
    stD = np.array(st)
    stD [stD == maskAction[act,1]] = maskAction[act,0]
    if sum(meta!=stD)==0:
        qBatchState.append(st)
        qBatchAction.append(np.array([ 0 if y !=maskAction[act,0]
                                      and y!=maskAction[act,1] else 1 for y in range(N)] ,
                                      dtype=np.int32))
        tamBatch[j] = 0
    else:
        validActionD = np.array(list(filter( lambda y: maskAction[y,0]
                                              in stD and maskAction[y,1] in stD,
                                              [x for x in range(len(maskAction))])))
        qBatchState.append(st)
        qBatchAction.append(np.array([ 0 if y !=maskAction[act,0]
                                      and y!=maskAction[act,1] else 1 for y in range(N)] ,
                                      dtype=np.int32))

        for w in validActionD:
            qBatchState.append(stD)
            qBatchAction.append(np.array([ 0 if y !=maskAction[w,0] and y!=maskAction[w,1]
                                         else 1 for y in range(N)] ,dtype=np.int32))
            tamBatch[j] = len(validActionD)
        qBatchState = np.array(qBatchState)
        qBatchAction = np.array(qBatchAction)
        bigQvals = np.array([x[0] for x in Qfun({'State':qBatchState,'Action':qBatchAction})))
        iniBigQvals=0
        # Fin obtencion de bigQvals
        for i,j in zip(indExp,range(len(indExp))):
            st = np.array(experience[i][0:N])
            act = int(experience[i][N])
            rew = experience[i][N+1]
            qval = bigQvals[iniBigQvals]
            iniBigQvals= iniBigQvals+1
            stD = np.array(st)
            stD [stD == maskAction[act,1]] = maskAction[act,0]
            if sum(meta!=stD)==0: # stD estado Final
                targ = rew
            else:
                qvalsD = bigQvals[iniBigQvals:(iniBigQvals+tamBatch[j])]
                iniBigQvals = iniBigQvals + tamBatch[j]

```



```

        bestScore = localScore
        bestCS = localCS
        lastCheck = ep
        Qfun.save("auxiliar_model.h5")

    else:
        if localCS < bestCS or ( localCS == bestCS and localScore <= bestScore) :
            if ep - lastCheck > patience:
                conv= True
            else :
                bestScore = localScore
                bestCS = localCS
                lastCheck = ep
                Qfun.save("auxiliar_model.h5")

    # Fin Revisar convergencia
    bar.update(ep)

# Termina Q - Learning
if check_convergence == True:
    Qfun = load_model("auxiliar_model.h5")
    B = self.buildAlignmentTree(sec,Qfun,maskAction)
    print( self.metrics(B))
    print("Tiempo ", time.time()-iniTiempo, " segundos")
    bar.finish()
    return B , Qfun , np.array(metricsGlobal)

def buildAlignment(self,Q,sec):
    if self.RLmethod == 'Tree':
        return self.buildAlignmentTree(Q,sec)
    if self.RLmethod == 'Permutation':
        return self.buildAlignmentPermutation(Q,sec)
    if self.RLmethod == 'Suffix':
        return self.buildAlignmentSuffixV2(Q,sec)

def align(self, sec, sizeExp=100, sizeMiniBatch = 100, M= 20, epsilon = 0.2,
         gamma = 0.99, alpha=0.1, patience=15, check_convergence=True):
    if self.RLmethod == 'Tree':
        return self.alignTree(sec, sizeExp, sizeMiniBatch, M, epsilon,
                             gamma, alpha, patience, check_convergence)
    if self.RLmethod == 'Permutation':
        return self.alignPermutation(sec, sizeExp, sizeMiniBatch, M, epsilon,
                                     gamma, alpha, patience, check_convergence)

local_alineador =AlineadorRL()

```

```

def generaEstado(longitudes):
    while True: ## Revisar no generar el estado final
        ans = np.array([np.random.randint(0,x+1) for x in longitudes])
        if sum(ans != longitudes) !=0:
            break
    return ans

def generaAccion(x,longitudes):
    while True: ## Revisar generar una accion valida para el estado en cuestion
        ans = np.array([np.random.randint(0,2) for x in longitudes])
        if sum(ans+x > longitudes) ==0:
            break
    return ans

def obtenRecompensa(st,act,datos):
    vec = "".join(['-' if act[i]==0 else datos[i][st[i]] for i in range(len(datos)) ] )
    #for x in datos: print(x)
    #print(st,act,vec)
    return local_alineador.SPvector(vec)

def generaBatch(BatchSize,datos):
    estadoInicial = np.array( [0 for x in datos])
    estadoFinal = np.array([len(x) for x in datos])
    # Generar batch aleatorio
    aux_state = [generaEstado(estadoFinal) for x in range(BatchSize)]
    aux_action = [generaAccion(x,estadoFinal) for x in aux_state]
    # Agregar estados "interesantes"
    ## 100 Estados Finales
    for w in range(100):
        while True:
            action = np.array([np.random.randint(0,2) for x in estadoFinal])
            if sum(action)>0:
                break
            aux_action.append(action)
            aux_state.append(estadoFinal-action)
    ## 100 Estados Iniciales
    for w in range(1000):
        aux_state.append(estadoInicial)
        aux_action.append(generaAccion(estadoInicial,estadoFinal))
    # Cargar datos
    estados = np.array(aux_state)
    acciones = np.array(aux_action)
    # Obtener recompensas

```

```

recompensas = np.array( [obtenRecompensa(st,act,datos) for st,act in zip(estados,acciones)])
return estados, acciones, recompensas

def create_modelTree(cant_sec):
    # Inputs
    inputAction = keras.Input(shape=(cant_sec,),name = 'Action')
    inputState = keras.Input(shape = (cant_sec,), name = 'State')

    # Proceso Action
    outAction = layers.Dense(16,activation="relu")(inputAction)

    # Proceso State
    emb1= layers.Embedding(input_dim=cant_sec, output_dim=5) (inputState)
    outState = layers.LSTM(16) (emb1)

    # Combinar
    comb = layers.concatenate([outAction,outState])
    combiD1= layers.Dense(16,activation="relu") (comb)
    outQ = layers.Dense(1) (combiD1)

    Q = keras.Model(inputs = [inputAction,inputState], outputs = [outQ])
    Q.compile(optimizer='adam',loss = 'mean_squared_error')
    return Q

def create_modelSuffix(cant_sec, long_sec):
    # Inputs
    inputAction = keras.Input(shape=(cant_sec,),name = 'Action')
    inputState = keras.Input(shape = (cant_sec,), name = 'State')

    # Proceso Action
    outAction = layers.Dense(16,activation="relu")(inputAction)

    # Proceso State
    emb1= layers.Embedding(input_dim=long_sec+1, output_dim=5) (inputState)
    outState = layers.LSTM(16) (emb1)

    # Combinar
    comb = layers.concatenate([outAction,outState])
    combiD1= layers.Dense(16,activation="relu") (comb)
    outQ = layers.Dense(1) (combiD1)

    Q = keras.Model(inputs = [inputAction,inputState], outputs = [outQ])
    Q.compile(optimizer='RMSprop',loss = 'mean_squared_error')
    return Q

def getQvals(estado,sec,maxlong,Qfun):
    #mat = alineador.getMatriz(estado,datos=sec, long=maxlong)
    N=len(sec)
    mascara= np.zeros((2**N-1,N),dtype=np.int32)
    for i in range(2**N-1):
        for j in range(N):
            if (i+1)&(1<<j)!=0:
                mascara[i,j]=np.int32(1)
            else:

```

```

mascara[i,j]=np.int32(0)
EvalState = []
EvalAction = []
for i in range(2**N-1):
    EvalState.append(estado)
    EvalAction.append(mascara[i])
Qvals = np.array(Qfun({"Action": np.array(EvalAction ),"State": np.array(EvalState )}))
Qvals = [x[0] for x in Qvals]
return np.array(Qvals)

def buildAlignmentSuffix(Qfun,sec):
    N=len(sec)
    mascara= np.zeros((2**N-1,N),dtype=np.int32)
    for i in range(2**N-1):
        for j in range(N):
            if (i+1)&(1<<j)!=0:
                mascara[i,j]=np.int32(1)
            else:
                mascara[i,j]=np.int32(0)
    meta = np.array([len(sec[x]) for x in range(len(sec))],dtype=np.int32)
    estadoActual = np.zeros((N))
    alin = []
    maxLong = np.max( [ len(x) for x in sec])
    while sum(estadoActual!=meta)!=0:
        Qvals = getQvals(estadoActual, sec, maxLong, Qfun)
        while True:
            a= int(np.argmax(Qvals))
            Qvals[a]=np.min(Qvals)-1
            if sum( (estadoActual+mascara[a])>meta) ==0: ## Ningun indice se debe exceder de la meta
                break
        estadoActual = estadoActual+mascara[a]
        alin.append([-1 if mascara[a][x]==0 else estadoActual[x]-1 for x in range(N)])
    alin=np.array(alin).T
    alineamiento = []
    for v,i in zip(alin,range(len(alin))):
        alineamiento.append("".join(['-' if x== -1 else sec[i][int(x)] for x in v]))
    return alineamiento

def NFQ(Niter,BatchSize,datos,gamma=1,epocas=10,patience=400):
    # Precalculos
    N=len(datos)
    mascara= np.zeros((2**N-1,N),dtype=np.int32)
    for i in range(2**N-1):
        for j in range(N):
            if (i+1)&(1<<j)!=0:

```

```

mascara[i,j]=np.int32(1)
else:
    mascara[i,j]=np.int32(0)
maxLong = np.max( [len(x) for x in datos] )
estadoFinal = np.array([len(x) for x in datos])

# NFQ
bar = progressbar.ProgressBar(maxval = Niter,widgets=[progressbar.Bar('=', '[', ']'),
                                                       ' ', progressbar.Percentage()])
bar.start()
metricas = []
primerCaso = True
i = 0
Q = create_modelSuffix(len(datos),maxLong )
iniT = time.time()
conv = False
while i < Niter and conv==False:
    # Obtener BATCH
    if i%50 == 0:
        batchState,batchAction, batchReward = generaBatch(BatchSize,datos)
        batchDestino = np.array([x+y for x,y in zip(batchState,batchAction)])
        batchDestinoQ = []
        batchActionQ = []
        for x in batchDestino:
            for y in mascara:
                batchDestinoQ.append(x)
                batchActionQ.append(y)
        batchDestinoQ = np.array(batchDestinoQ)
        batchActionQ = np.array(batchActionQ)

    # Iteracion
    QvalsT = np.array(Q({'Action': batchActionQ, 'State': batchDestinoQ}))
    QvalsT = np.array( [x[0] for x in QvalsT])
    QvalsT = QvalsT.reshape((len(batchDestino), len(mascara)))
    batchTarget = []
    for st, act , rew, stNext, Qvals in zip(batchState,batchAction,batchReward,batchDestino,QvalsT):
        stNext = st + act
        #
        if sum(stNext != estadoFinal)==0:
            q_max=0
        else:
            while True: # Elegir q_max de entre las acciones validas
                a_aux = int(np.argmax(Qvals))
                q_max = np.max(Qvals)
                Qvals[a_aux] = np.min(Qvals)-1
                if sum( (stNext + mascara[a_aux])>estadoFinal) == 0:

```

```

        break

target = rew + gamma * q_max
batchTarget.append(target)
batchTarget = np.array(batchTarget)

Q.fit({"Action": batchAction, "State": batchState}, y=batchTarget, epochs=epocas,
       verbose=0, batch_size=BatchSize)

B = buildAlignmentSuffix(Q, datos)
met = local_alineador.metricas(B)
metricas.append(met)

if primerCaso:
    bestSP = met[0]
    bestCS = met[3]
    bestIt = i
    Q.save("auxiliar_model.h5")
    primerCaso = False
else:
    if met[0]>bestSP or (met[0]==bestSP and met[3]>bestCS):
        bestSP = met[0]
        bestCS = met[3]
        bestIt = i
        Q.save("auxiliar_model.h5")
    else:
        if i - bestIt>patience:
            conv = True
i = i+1
bar.update(i)

Q = load_model("auxiliar_model.h5")
B = buildAlignmentSuffix(Q, datos)
print(local_alineador.metricas(B))
print("Tiempo :", time.time() - init)
bar.finish()

return B, Q, np.array(metricas), bestIt

```