Díaz, Adrián Gustavo

Efectividad del método de búsqueda metaheurístico enfriamiento simulado en la determinación de una secuencia consenso a partir de múltiples secuencias

Adrián Gustavo Díaz, Repositorio Institucional Digital de Acceso Abierto,
Octubre de 2024, pp. 111,
http://ridaa.unq.edu.ar,
Universidad Nacional de Quilmes, Secretaría de Posgrado,
Maestría en Bioinformática y Biología en Sistemas

Efectividad del método de búsqueda metaheurístico enfriamiento simulado en la determinación de una secuencia consenso a partir de múltiples secuencias

TESIS DE MAESTRÌA

Adrián Gustavo Díaz
diaz.adrian.g@gmail.com

Resumen
La evolución del estudio de la biología

Uno de los eventos que cambia la forma de estudiar y hacer Biología es la capacidad de estudiar los lenguajes que escriben las componentes de la vida. Una vez que comprendimos y entendimos que la información genética está almacenada en las cadenas de ADN a principios de la década de 1950, la velocidad de nuevos descubrimientos avanzó a una velocidad exponencial. A finales de la década de 1970, el científico Sanger pudo secuenciar un fragmento de ADN, abriendo la puerta a conocer el orden de nucleótidos que conforman esa sección de la molécula. Con el paso del tiempo, los métodos fueron mejorando tanto en velocidad como en costos por lectura. La secuenciación, es decir, la lectura y representación de un gen, ARN mensajero o proteína (entre otros) dio paso a una nueva etapa en la Biología.

Es inevitable que la Informática se vuelva parte central de los procesos de investigación en la ciencia, y la Biología no es la excepción. De hecho, la Bioinformática surgió como una ciencia que combina entre otras a la Informática con la Biología en pos de extraer información y nuevo conocimiento desde todos los datos extraídos de las muestras biológicas. Este trabajo muestra, a lo largo de su desarrollo, un claro ejemplo de la contribución de la Computación en la Biología.

Actualmente hay una cantidad incalculable de datos por ser estudiada, procesada y usada como fuente de entrada de nuevos procesos de investigación, desarrollo y aplicación. En el caso de la secuenciación de entidades biológicas como ADN de genes, ARN mensajeros, aminoácidos de secuencias proteínas, por no hablar de artefactos completos como genomas, proteomas, transcriptomas, entre otros. El análisis de dichas entidades en conjunto permitió caracterizar al conjunto más allá del individuo. Entre los muchos ejemplos,

se pueden deducir regiones que han sido conservadas a lo largo de los procesos evolutivos de las diferentes especies estudia-das. El éxito de estos estudios bioinformáticas se ve plasmado en que sea cotidiano realizar alineamientos múltiples de secuencias o caracterizar una familia de proteínas con una secuencia consenso o un logo. Todo esto es posible gracias al trabajo multidisciplinario entre expertos biólogos e informáticos desde hace muchos años. La idea del autor al momento de efectuar este trabajo es aportar una comparación entre herramientas existentes en el ámbito de la construcción de alineamientos múltiples de secuencias y una implementación de la metaheurística “Enfriamiento Simulado” (SA por sus siglas en inglés de Simulated Annealing). Este trabajo describe el camino recorrido al crear un método bioinformática y sentar las bases de comparación y evaluación de utilidad para futuros investigadores. Tanto el código de la implementación SA como los protocolos de comparación y evaluación se encuentran disponibles en un repositorio de código presentado más adelante en este documento con la finalidad de ser consultados.

Maestría en Bioinformática y Biología de Sistemas

Trabajo Final de maestría

Efectividad del método de búsqueda metaheurístico enfriamiento simulado en la determinación de una secuencia consenso a partir de múltiples secuencias

Aspirante: Ing. Adrián G. Díaz
Directora: Dra. Gabriela Minetti
Abril 2024

Índice de contenidos

Motivación e hipótesis

La Biología encontró en la Informática un aliado para ir más allá de sus métodos de observación y estudio clásicos. Esta unión resultó en la aparición de la Bioinformática, un campo de la ciencia relativamente nuevo que de la mano de los avances de los equipos de secuenciación masiva, entre otras invenciones, técnicas y desarrollos, provee a los investigadores biológicos de información en cantidades inimaginables mediante diversos algoritmos implementados en aplicaciones informáticas [1].

La aparición de métodos de secuenciación masiva a fines del siglo pasado, junto al trabajo en conjunto de diversos laboratorios y empresas privadas enfocados al mismo tiempo en crear los métodos con el fin de interpretar los datos, han contribuido a que hoy en día existan miles de bases de datos de información biológica además de centenares de herramientas de *software* disponibles para cualquier investigador.

Sin dudas, contar con el conocimiento de la información inherente a una célula permite expandir los límites de las ciencias de la vida. Esta búsqueda comenzó a mitad del siglo XX cuando se descubrió que la información genética es almacenada en una estructura de doble hélice denominada Ácido Desoxirribonucleico, más conocida pos sus siglas ADN. Con el pasar de los años se desarrollaron métodos de laboratorio con el objetivo de determinar la secuencia de nucleótidos que conforman una secuencia de ADN. Hacia fines del siglo pasado, se entró en una nueva era a causa del surgimiento de nuevos secuenciadores con un poder de procesamiento industrial. Llegando, en el siglo XXI, a la publicación del genoma humano luego del esfuerzo de diversos consorcios científicas y empresas de Biotecnología [2].

Hoy en día, trabajar con bases de datos y técnicas bioinformáticas está asimilado en los grupos de investigación en Biología. El estudio en simultáneo de varias secuencias relacionadas como un todo es la puerta de entrada a estudios más complejos. Este tipo de análisis es conocido como Alineamiento Múltiple de Secuencias, abreviado generalmente como *MSA* (por sus siglas en inglés *Multiple Sequence Alignment*).

Dada la naturaleza evolutiva de la vida, siendo un axioma fundamental de la biología que conocemos en nuestro planeta, conocer qué regiones son conservadas entre distintas muestras de secuencias es vital para convertir los datos embebidos en las secuencias en información biológica [3]. Una vez conocidas, entender el por qué de su conservación permite hallar las piezas claves en relación al funcionamiento de los individuos estudiados. Estas investigaciones sobre la conservación y variabilidad abren paso a la determinación de la historia de las especies

mediante árboles filogenéticos, a obtener patrones de homología que luego se usan con la finalidad de modelar estructuras, como así también, la de buscar secuencias biológicamente similares a partir de una de interés, por nombrar algunos ejemplos. Es así que dentro del análisis de patrones presentes en múltiples secuencias, existe el concepto de secuencia consenso o canónica. Dicha secuencia representa al nucleótido o amino ácido (en el caso de proteínas) más frecuente para cada posición dentro de la múltiples secuencias estudiadas.

Dado que encontrar un MSA es un problema que computacionalmente no puede resolverse en un tiempo polimonial [4], surgen las siguientes hipótesis:

1. **H1**: Los algoritmos aproximados, como las metaheurísticas, basados en Enfriamiento Simulado o SA (por sus siglas en inglés *Simulated Annealing*) resuelven eficientemente el MSA [5].

2. **H2**: Estos algoritmos son competitivos con el estado del arte a la hora de resolver el MSA.

3. **H3**: Es posible combinar o hibridar diferentes funciones objetivo para encontrar un MSA suficientemente bueno en comparación con las otras alternativas en el estado del arte.

4. **H4**: Se puede crear un marco de trabajo para comparar diferentes *software* MSA usando métricas del estado del arte para ordenarlos por calidad de los resultados.

Siguiendo el método científico, este trabajo de maestría es un estudio comparativo de *software* de MSA representativos del estado del arte de la Bioinformática y una implementación del método metaheurístico SA. Para demostrar o refutar las hipótesis se realiza la comparación y evaluación de los *software* elegidos con grupos de secuencias homólogas estandarizadas, donde ya se conoce su alineamiento óptimo. La condición de evaluación es en base a diferentes métricas ya utilizadas por la comunidad científica.

Objetivos y fases

Para poder llevar a cabo este planteo es necesario cumplir con los siguientes objetivos:

- Implementar un algoritmo de búsqueda metaheurístico basado en Enfriamiento Simulado (*SA* por sus siglas en inglés) para el problema de hallar un MSA y combinarlo (hibridarlo) con heurísticas y así mejorar su desempeño.

- Realizar un estudio comparativo de diferentes *software* MSA, eligiendo a los más representativos de cada tipo. Dicho desempeño está atado a la función objetivo utilizada para medir la calidad de un MSA.

    - Desarrollar un marco de trabajo suficientemente flexible para evaluar múltiples *software* MSA a través de diferentes métricas de calidad.
    - Evaluar utilizando el uso del marco de trabajo del objetivo anterior, si el algoritmo propuesto es lo suficientemente bueno en comparación a las aplicaciones existentes actualmente mediante la comparación de similitud con el resultado patrón (definido de antemano por los curadores de las bases de datos de pruebas).

Organización del documento

Este documento está organizado en capítulos que siguen temporalmente la investigación, desarrollo y análisis efectuados:

1. **Contexto Bioinformático**: Partiendo de una resumida historia de la historia de la Bioinformática se llega a la importancia del estudio en conjunto de secuencias. Una vez explicada la importancia, se dan a conocer los detalles sobre cómo es que los objetivos de este trabajo son desarrollados.

2. **Enfoque bioinformático de MSA**: Contiene el contexto bioinformático de este tipo de estudio.

3. **Implementación de SA para el problema MSA**: Explica las particularidades de la implementación SA para el problema de alinear múltiples secuencias. Se explica en detalle la construcción del algoritmo MSASA.

4. **Diseño experimental**: Contiene las especificaciones de todos los pasos que se realizan para validar la hipótesis. Es decir, abarca desde la selección de las secuencias de prueba, el *software* MSA a comparar y los pasos de evaluación de lo resultados.

5. **Análisis de los resultados**: Aquí se trabaja sobre los resultados encontrados una vez comparadas las herramientas bioinformáticas del estado de arte y MSASA.

6. **Debate y conclusiones**: Se debate la validez de la hipótesis a partir de los resultados analizados en la sección anterior.

7. **Bibliografía**: Contiene las citaciones utilizadas a lo largo de todo el trabajo práctico.

8. **Anexos**: Adentro de estas páginas se detallan los pormenores a nivel técnico de la implementación MSASA.

1. La Bioinformática y el Alineamiento Múltiple de Secuencias

La Bioinformática es un campo interdisciplinar que involucra a los expertos en Biología Molecular, matemáticos, ingenieros y físicos para: alinear secuencias, analizar los genes, identificar y predecir las estructuras moleculares, determinar el perfil de expresión genética. Esta disciplina incluye técnicas computacionales y aplicaciones que llevan a cabo esas actividades, siendo enorme el número de estas técnicas y aplicaciones y muy significativas las diferencias entre ellas. En este capítulo se explican los conceptos relacionados con la Biología celular y molecular, los hitos de la Bioinformática para poder describir algunas de las herramientas más importantes de esta ciencia, como son las secuencias consenso y los alineamientos múltiple de secuencias.

1.1. Breves conceptos de la Biología celular y molecular

1.1.1. La célula y las macromoléculas de la vida

Todos los seres vivos tienen en común que su unidad básica es la célula, algunos organismos son unicelulares, como por ejemplo las bacteria, mientras que otros son compuestos una gran cantidad de estos componentes básicos, por ejemplo, los animales. Recibe el título de unidad porque es la entidad más pequeña que puede vivir por sí misma y es capaz de brindar las funciones más básicas de la vida: darle un cuerpo, obtener energía y utilizarla con el objeto de realizar diferentes actividades a través de reacciones químicas [6].

Todas las células tienen una capa que las rodea llamada membrana celular, y es por medio de ella que una célula puede interactuar con otros componentes. Dentro de la célula se encuentra el núcleo donde se haya el materia hereditario del organismo que es pasado a los descendientes. Algunas células son más complejas y tienen una capa que envuelve al núcleo, este tipo es conocido como células eucariotas. Al contrario, las células que no poseen esta capa extra son conocidas como procariotas. Todas las bacterias y arqueas son seres unicelulares procariotas. Por último la célula contiene un líquido en su interior llamado citoplasma donde se alojan una serie de organelos encargados de funciones específicas de la vida celular.

El ADN, o ácido desoxirribonucleico, es una macromolécula de la vida en forma de doble hélice. Esta contiene la información hereditaria de los organismos vivos. La información en el ADN se determina mediante un código compuesto por cuatro bases química donde el orden de

estas bases determina la información para construir y mantener un organismo. El ADN tiene la capacidad de replicarse y hacer copias de sí mismo dado que cada hebra de ADN puede servir como patrón para duplicar la secuencia de bases.

Un gen es la unidad básica de la herencia y están formados por ADN. La mayoría de los genes contienen la información necesaria para producir moléculas funcionales llamadas proteínas y algunos otros codifican otro tipo de moléculas. El flujo de información del ADN a las proteínas es uno de los principios fundamentales de la biología molecular.

Las proteínas, o cadenas de polipéptidos, son moléculas que desempeñan muchas funciones en los organismos. Realizan la mayor parte del trabajo en las células y son necesarias para la estructura, función y regulación de los tejidos y órganos. Las proteínas están formadas por unidades conocidas como aminoácidos o residuos, que se unen entre sí en largas cadenas. En la naturaleza hay 20 tipos diferentes de aminoácidos que se pueden combinar para formar una proteína. La secuencia de aminoácidos determina la estructura tridimensional única de cada proteína y su función específica.

1.1.2. El descubrimiento de la estructura del ADN

En abril de 1953 se publicó el artículo que proponía la estructura de doble hélice del ADN en la revista Nature, escrito por el biólogo James Watson y el biofísico Francis Crick [7], ambos trabajando en el laboratorio *Cavendish* de la Universidad de Cambridge. El impacto fue de tan magnitud que revoluciono la biología y sentó las bases fundamentales para el desarrollo del "Dogma de la biología molecular".

Su obra estuvo basada en los estudios de cristalografía realizados por Maurice Wilkins y Rosalin Franklin donde se podía intuir que el ADN tiene un estructura helicoidal, además de las observaciones de otros científicos que notaron como las proporciones de bases de nitrogenadas se presentaban en igual forma entre Adenina-Timina y Citosina-Guanina. La cristalógrafa Franklin trabajaba en la Unidad de Biofísica del Consejo de investigación Médica del *King's College London* cuando logró distinguir una forma cristalina A y una forma hidratada B del ADN.

Gracias a las fotografías obtenidas por ella, Watson y Crick pudieron ver como la simetría de la forma A cristalina indicaba que las cadenas de azúcar-fosfato debían ser antiparalelas, de tal manera que las bases nitrogenadas encajan en una estructura cilíndrica formada por la doble hélice, emparejando las bases del tipo purina con las del tipo pirimidina mediante puentes de hidrógeno (dos puentes de hidrógeno en las bases enfrentadas de Adenina y Timina, o tres puen-

tes en las bases Guanina y Citosina). Por esta razón, se dice que las hebras son complementarias a causa de que este emparejamiento de bases implica que el orden de nucleótidos de una de las dos hebras determina el orden de la otra. [8]

Crick postuló luego del descubrimiento de la estructura física del ADN una serie de estados posibles de transferencia de la información biológica entre los diferentes polímeros de ADN, ARN y proteínas [9]. Si bien con el paso del tiempo se dio un debate sobre si este modelo era una sobre-simplificación o no, sentó las bases para el entendimiento a nivel molecular de la transmisión secuencial de información y con el avance de la tecnología, se pudo distinguir entre probables, posibles o imposibles transferencias.

Como se menciono anteriormente, el ADN almacena la información hereditaria dentro de compartimientos llamados genes. La vida evolucionó para crear un lenguaje de solo cuatro bases nitrogenadas con el cual se puede codificar la información genética, dichas bases son la Adenina (A), Citosina (C), Guanina (G) y la Timina (T), que junto a una pentosa y un grupo fosfato forman una base química llamada nucleótido. La molécula de ADN es una secuencia de nucleótidos dentro de una estructura de dos hebras forma de espiral conocida como doble hélice, donde las bases se emparejan entre sí (Adenina sólo con Timina y viceversa, o Citosina con Guanina o viceversa) dando lugar a los llamados “pares de bases”. Mientras tanto, los azúcares y fosfatos hacen de soporte o esqueleto de la estructura de doble hélice.

En la publicación original (figura 1), se explica como la información codificada en las moléculas de ADN es usada con el fin de generar las proteínas: “Existe una relación entre la ordenación lineal de los nucleótidos en el ADN y la ordenación lineal de los aminoácidos en las proteínas”. Conjuntamente, se postuló que la información genética, está almacenada únicamente en el ADN y este puede tanto duplicarse como pasar a los descendientes de un ser vivo.

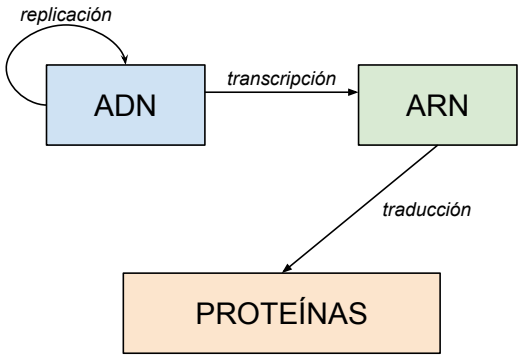

Figura 1: Esquema básico del Dogma fundamental donde las lineas continuas reflejan el flujo general de la información biológica y las líneas punteadas representan situaciones excepcionales descubiertas a posteriori.

1.1.3. El Dogma fundamental de la Biología

Este postula que la información en los organismos vivos se transmite desde el ADN a través del ARN hasta las proteínas, un proceso que consiste en dos etapas principales: la transcripción y la traducción. En la transcripción, la información del ADN se copia en una molécula de ARN mensajero (ARNm). Luego, en la traducción, el ARNm dirige la síntesis de proteínas. En esencia, el dogma sugiere que "el ADN produce ARN y el ARN produce proteínas" [10]. Sin embargo, los avances científicos han descubierto excepciones a este dogma (figura 2).

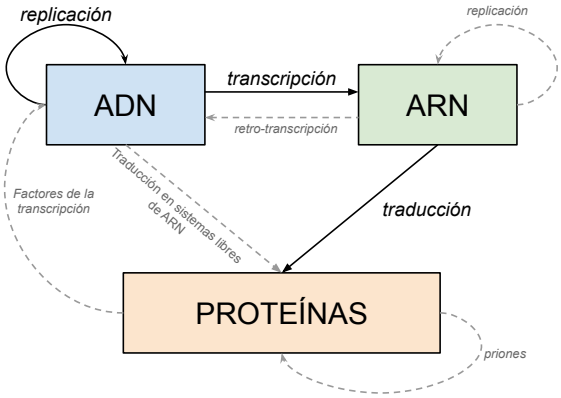

Figura 2: Esquema básico del Dogma fundamental donde las lineas continuas reflejan el flujo general de la información biológica y las líneas punteadas representan situaciones excepcionales descubiertas posteriormente.

Mediante el orden de los pares de base de las moléculas de ADN es posible confeccionar los genes que darán lugar a otras moléculas responsables del construir y dar funcionamiento a un ser vivo. El gen es la mínima unidad y está alineado en ubicaciones especificas dentro del ADN de cada individuo. Cada especie cuenta con un genoma (colección de genes) específico,

y luego cada individuo cuenta con pequeñas variaciones que lo hacen único.

En oposición a lo que se podría pensar, la proporción de genes o regiones que crean proteínas es muy baja en relación a las regiones que no contienen instrucciones para generar proteínas. Con el tiempo, se han ido explicando las funciones de estas regiones no codificantes, pudiendo mencionar sitios para regular los procesos de expresión de proteínas (promotores o represores), instrucciones para generar moléculas de ARN de transferencia y ribosómico (de gran ayuda para la construcción de proteínas).

El ADN es capaz de replicarse y así generar una copia de sí mismo en lo que se conoce como el proceso de *replicación* usándose a si mismo como patrón para obtener dos copias idénticas, o al menos con la menor cantidad de errores posible generados por mutaciones que pueden alterar la información original.

Partiendo del ADN es posible construir proteínas mediante el proceso de transcripción. Al leer genes codificantes de proteínas, se transforma la información codificada en los nucleótidos del ADN en nuevas moléculas similares pero de una sola hebra llamadas *ARN mensajero* (conocidos por su sigla ARNm). Este polímero luego por otro proceso de maduración antes de estar listo y sea interpretado por las moléculas encargadas de producir proteínas.

La columna vertebral de una proteína está formada por el enlace peptídico, una repetición de los patrones de átomos de Nitrógeno-Carbono-Carbono. Las cadenas laterales, también conocidas como grupos R, son diferentes para cada aminoácido y se unen a los carbonos alfa de la columna vertebral. La naturaleza química (hidrofílica o hidrofóbica, por ejemplo) de estas cadenas laterales influirá en cómo se pliega la proteína y, en última instancia, determinará su función.

En términos de estructura química, un aminoácido se compone de un grupo carboxilo, un grupo amino, un átomo de hidrógeno y una cadena lateral (o grupo R) que varía entre los diferentes aminoácidos. La secuencia y el número de aminoácidos en última instancia determinan la forma y la función de una proteína.

Existen 20 aminoácidos estándar que forman parte de la maquinaria genética de casi todos los organismos, pero también existen algunos aminoácidos no estándar que son menos comunes y a menudo se encuentran en casos especiales o en ciertos tipos de organismos [11].

Las proteínas son polímeros complejos de vital importancia en las células ya que realizan diferentes funciones dentro de las células, o pueden ser parte de su estructura, o ser envidas al exterior para funcionar como una molécula señalizadora. Como ejemplo de funciones realizadas por las proteínas se encuentran:

- **Anticuerpos**: son proteínas que se unen a partículas extrañas especificas para proteger al individuo.
- **Enzima**: son las proteínas que se encargan de llevar a cabo las reacciones químicas que ocurren en las células.
- **Mensajeros**: transmiten señales para coordinar procesos biológicos entre distintos entes como células, tejidos u órganos.
- **Estructural**: son las proteínas que dan estructura y suporte a la célula.
- **Transporte**: estas proteínas se unen a ciertos átomos o moléculas pequeñas para movilizarlas dentro de la célula o por el cuerpo.

En general, la función que tiene una proteína viene dada por la forma que tiene al plegarse y obtener su estructura en tres dimensiones. Este plegamiento se da gracias a diferentes fuerzas de interacción que aparecen entre los aminoácidos que la forman. Las proteínas están organizadas y estudiadas en diferentes niveles de estructura. La primaria es directamente la secuencia de aminoácidos. Al plegarse en en el plano tridimensional se forman diferentes estructuras conocidas, como el hélice alfa (en inglés *Alpha-Helix*) o la hoja plegada (en inglés *Beta-Sheet*). A este nivel se lo conoce como estructura secundaria de la proteína. Una vez que esta termina de plegarse sobre si misma, se obtiene la estructura terciaria. Puede existir el caso donde diferentes proteínas formen una estructura juntas, en ese caso se habla de estructura cuaternaria (figura 3).

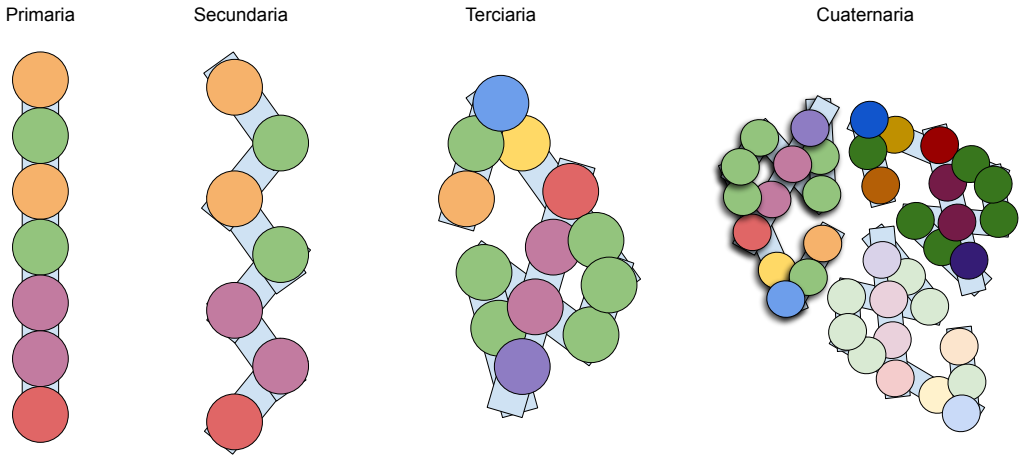

Figura 3: Los diferentes niveles de estructuras de una proteína

Las proteínas son creadas dentro de la célula mediante unas estructuras llamadas ribosomas durante el proceso de traducción. Son responsables de la síntesis de proteínas a través de

un proceso conocido como traducción. Durante la traducción, los ribosomas decodifican la información genética contenida en el ARN mensajero (ARNm) para ensamblar las cadenas de residuos que luego se pliegan en proteínas funcionales.

El código genético es el sistema que permite que la información genética almacenada en el ADN y el ARN sea traducida en proteínas en las células vivas. Es esencialmente un conjunto de reglas que define cómo se traduce una secuencia de nucleótidos en una secuencia de aminoácidos.

Un codón es una secuencia de tres nucleótidos en el ARN que corresponde a un aminoácido en una proteína. En el ADN, la secuencia equivalente se llama triplete. Los codones son la unidad fundamental del código genético; cada uno actúa como una instrucción para añadir un aminoácido específico a una cadena en crecimiento de una proteína durante el proceso de traducción. La ribosoma lee el ARN mensajero secuencialmente de a tres bases por vez para ir ensamblando de a un aminoácido por triplete. Este mecanismo de ensamble de aminoácidos continua hasta leer el triplete de parada [12] (tabla 1).

El código genético es universal entre todos los organismos conocidos, con algunas excepciones menores, lo que significa que el mismo codón en cualquier organismo generalmente codifica el mismo aminoácido. Por ejemplo, el codón *AUG* siempre codifica para el aminoácido Metionina, y también actúa como el codón de inicio para la traducción. Hay tres codones de parada (*UAA*, *UAG*, *UGA* en ARN y *TAA*, *TAG*, *TGA* en ADN) que señalan el fin de la traducción. Aunque existen algunas excepciones donde el código genético no es el estándar.

Las proteínas están formadas por 20 aminoácidos diferentes y, como hay 64 posibles codones (4 nucleótidos posibles en cada una de las 3 posiciones del codón), más de un codón puede codificar para un aminoácido dado. Esto se conoce como degeneración del código genéticos.

	U	C	A
UU	UUU Fenilalanina (Phe)	UCU Serina (Ser)	UAU Tirosina (Tyr)
CU	CUU Leucina (Leu)	CCU Prolina (Pro)	CAU Histidina (His)
AU	AUU Isoleucina (Ile)	ACU Treonina (Thr)	AAU Asparagina (Asn)
GU	GUU Valina (Val)	GCU Alanina (Ala)	GAU Ácido aspártico (Asp)

Tabla 1: Ejemplo de la tabla del código genético de triple entrada. Las filas representan los primeros dos nucleótidos del codón del ARN, y las columnas representan el tercer nucleótido del codón. Cada celda está subdividida en cuatro sub-celdas que representan los diferentes codones y sus correspondientes aminoácidos.

1.2. Hitos en la Bioinformática

1.2.1. La evolución de las técnicas de secuenciación

Antiguamente, para conocer la estructura de una proteína se la fragmentaba en pequeñas piezas con el fin de saber su composición y luego intentar deducir el orden de los aminoácidos a partir de los fragmentos que se solapaban entre si. A finales de la década de 1970 aparecieron los primeros desarrollos científicos y tecnológicos que permitieron a la comunidad determinar la secuencia de nucleótidos de un fragmento cualquiera de ADN basados en los métodos conocidos para proteínas pero con la gran limitación de que en las proteínas el alfabeto es de veinte aminoácidos posibles mientras que en el dominio del ADN solo hay cuatro posibles valores, lo cual, complicaba encontrar los solapamientos correctos.

Las técnicas de secuenciación fueron beneficiadas por el descubrimiento de la técnica de *Reacción en cadena de la Polimerasa* (conocido por su sigla en inglés *PCR*) que permitió obtener miles de copias de un fragmento de ADN, dando lugar a mejorar la calidad de las muestras biológicas [13].

Los dos protocolos a principios de 1980 se basaban en marcar las moléculas a secuenciar mediante algún método radiactivo o fluorescente, para luego efectuar la separación de la muestra en cadenas que difieren en tamaño por una base. Estas cadenas de diferentes largos pueden separarse mediante técnicas de electroforesis en geles desnaturalizantes permitiendo a los científicos

interpretar las bandas que se forman y de esa manera conocer la secuencia de nucleótidos del fragmento estudiado.

1.2.2. El Método de Secuenciación Sanger, o método patrón de oro

En la actualidad este es el protocolo que se utiliza cuando hay alguna duda o se requiere una gran fiabilidad. El procedimiento fue diseñado por Sanger, Nicklen y Coulson en 1977 bajo el nombre de *método de terminadores de cadena o dideoxi* [14]. Durante la secuenciación se efectúan cuatro síntesis separadas, utilizando pequeñas cantidades de dideoxinucleótidos (ddGTP, ddATP, ddCTP, ddTTP) que no tienen el extremo 3' OH libre y al ser añadidas a la secuencia sintetizada hacen que se termine abruptamente su elongación. Sustituyendo los dideoxinucleótidos por el mismo nucleótido marcado en forma radioactiva permite a los científicos ver las bandas de distinta longitud en el gel y así conocer hasta trescientas bases por estudio.

1.2.3. Las técnicas de secuenciación masiva y su impacto en la Biología

Con el pasar de los años, las mejoras hechas sobre la secuenciación tradicional del método Sanger han logrado bajar el costo por base secuenciada aproximadamente a una centésima parte del costo original [15]. Si bien hay en el mercado una cantidad considerable de empresas y modelos disponibles, en general todas siguen un protocolo común: confeccionar una librería de un tamaño determinado, hacer una amplificación clonal, dar inicio a varios ciclos de secuenciación masiva en paralelo que luego serán procesados mediante herramientas de análisis bioinformáticos con el objetivo de interpretar los datos. Dependiendo de el equipamiento utilizado, y su estrategia al implementar el protocolo mencionado, será la calidad y cantidad de bases secuenciadas. Estos equipos de secuenciación generan archivos con el contenido de las muestras y son conocidos como lecturas. Hay dispositivos que devuelven lecturas largas y otros cortas, pero al fin y al cabo, el *software* bioinformático tienen la tarea de unir estas lecturas para producir la secuencia completa de las muestras introducidas en el secuenciador. Las nuevas técnicas de secuenciación de alto rendimiento, conocidas por su sigla *HTS* (de la expresión en inglés *High-Throughput Sequencing*), pueden producir hasta mil millones de lecturas por vez, lo cual abre un abanico de posibilidades para la investigación biológica. Incluso la capacidad de secuenciación no sigue la Ley de Moore, dado que se duplica de seis a nueve meses en lugar de los dos años que tardan los microprocesadores [16].

Es imposible no observar como estas nuevas tecnologías cambiaron la forma de hacer biología, posibilitando a los científicos estudiar genomas, proteomas, transcriptomas de individuos,

o metagenomas de muestras más generales, siempre de la mano de la bioinformática, pudiendo entre otros ejemplos hacer una comparación de genes o proteínas entre diversos individuos para encontrar patrones y variantes (mutaciones)[17]. Esto último se hace mapeando las lecturas sobre un genoma o proteoma de referencia para luego utilizar *software* bioinformático que buscan las diferencias.

Es por todo esto que dentro de la Bioinformática, el alineamiento múltiple de secuencias o la cadena consenso, son herramientas de base para cualquier otro estudio que esté relacionado con la evolución y conservación de la información biológica.

Adicionalmente, apareció otra técnica visual basada en logos para representar un MSA, donde en cada posición se modela la proporción de elementos posibles a través de la altura de los mismos. A mayor altura, mayor es la probabilidad de que ese elemento sea encontrado en tal posición, y viceversa para alturas por debajo del eje de las abscisas (eje X) [18].

1.3. Las Secuencias Consenso

Una secuencia consenso es una representación "promedio" de un conjunto de secuencias la cual reflejaría la secuencia más común del conjunto. Se utilizan a menudo en la identificación de motivos conservados en las secuencias de ADN, ARN o proteínas, lo que puede proporcionar las funciones biológicas de estas secuencias y la evolución entre las diferentes secuencias estudiadas.

Estas secuencias consenso también pueden ayudar a identificar sitios de unión para proteínas y otras moléculas, regiones de codificación genética y segmentos de ADN o ARN que se conservan en diferentes especies o en diferentes genes dentro de una especie.

Considerando tres secuencias de ADN de cinco bases de longitud de distintas especies para el mismo gen:

- **Secuencia Gen A - Especie 1**: A T G C A
- **Secuencia Gen A - Especie 2**: A T G C G
- **Secuencia Gen A - Especie 3**: A T G C T

En este caso, la cadena consenso sería "ATGCA", ya que el primer, segundo, tercer y cuarto nucleótidos son idénticos en todas las secuencias. En la quinta posición, hay una diferencia, pero el nucleótido "A" es el más común, por lo que se elige para la cadena consenso (tabla 2).

Posición	Gen Especie 1	Gen Especie 2	Gen Especie 3	Secuencia Consenso
1	A	A	A	A
2	T	T	T	T
3	G	G	G	G
4	C	C	C	C
5	A	G	T	A

Tabla 2: Ejemplo de generación de una cadena consenso a partir de tres secuencias de ADN.

1.4. El Alineamiento Múltiple de Secuencias (MSA)

Alinear tres o más secuencias de nucleótidos o amino ácidos es una de las tareas más comunes, pero a la vez importantes para los bioinformáticos. Su resultado es el punto de partida de muchos otros análisis biológicos o bioinformáticos debido a que más cercanas sean dos macromoléculas en términos evolutivos, más similares y conservadas serán sus secuencias. El resultado de dicho alineamiento es utilizado para diversos propósitos dentro del secuenciamiento genómico, el modelado estructural, determinar familias de proteínas homologas (comparten un ancestro común), reconstruir las relaciones filogenéticas entre las secuencias, o incluso anotar las regiones conservadas que han tenido baja variabilidad a lo largo de la evolución en el tiempo. [19]

Cuando se hace un alineamiento de un par de secuencias, se busca maximizar una puntuación que refleje las características compartidas de las secuencias, es decir que haya una mayor coincidencia de bases entre ambas secuencias: si ambas secuencias son evolutivamente cercanas, una no-coincidencia puede ser interpretada como una mutación, mientras que un espacio o *gap* se considera como una inserción/deleción, pues ambas secuencias divergen a través del tiempo. Las regiones compartidas dan un indicio de que esos nucleótidos o aminoácidos tienen una importancia vital para el funcionamiento correcto de las secuencias y una mutación en esa región puede desencadenar que el individuo no logre trascender a lo largo del tiempo.

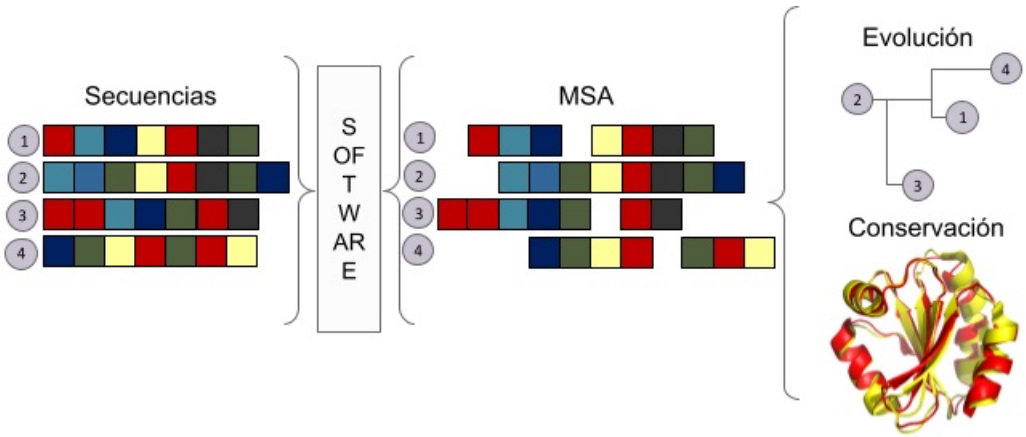

Figura 4: A partir de un grupo de tres o más secuencias, se puede generar un MSA a través de un *software* bioinformático que es clave en el estudio de la información biológica del grupo, como por ejemplo, en el entendimiento de la evolución y conservación de regiones comunes entre las secuencias.

La realización de MSA también es esencial para la construcción de secuencias consenso. A partir de un MSA, se puede generar una secuencia consenso que represente las características comunes de tres o más secuencias alineadas. Esto puede ayudar a identificar regiones conservadas en las secuencias, que a menudo son de interés biológico. Además, las secuencias consenso generadas a partir de MSA pueden usarse para buscar en bases de datos de secuencias biológicas y encontrar secuencias adicionales que tengan características similares [20].

1.4.1. Definición formal de MSA

Citando la definición formal escrita por Srinivas Aluru [21], una alineación múltiple A del conjunto de secuencias S_1, ..., S_k, donde k es mayor o igual que dos secuencias, se obtiene insertando espacios en cada cadena para obtener k cadenas S_1', ..., S_k' de la misma longitud. Para cada par de i, $j \in [k]$, $i \neq j$, la alineación A induce una alineación por pares A_{ij} entre las cadenas S_i y S_j, obtenida tomando S_i' y S_j' y eliminando cualquier columna que contenga dos espacios. Supongamos que hemos elegido algún mecanismo de puntuación para alineaciones por pares que depende de un vector de parámetros λ. Entonces, la puntuación de suma de pares (SP) de A se da por:

$$score(A,\lambda) = \sum_{i<j} score(A_{ij},\lambda)$$

1.4.2. Sobre la calidad de un MSA

Las secuencias en un MSA cuyo último ancestro común es mucho más evolutivamente distante que el de las secuencias restantes en el alineamiento se pueden denotar como secuencias divergentes. Este escenario también puede surgir como resultado de procesos de transferencia horizontal de genes y también a través de errores y fallos durante las diferentes fases que conducen a un MSA.

Las brechas o 'gaps' en un MSA pueden resultar de la adquisición o pérdida de información biológica representada por residuos de nucleótidos o aminoácidos. Ocasionalmente, es difícil identificar la verdadera causa entre estas dos posibilidades, pero en cualquier caso, para mantener la similitud (u homología) en áreas no totalmente conservadas, es necesario insertar brechas. Además, los métodos actuales para construir MSAs no son perfectos y a menudo introducen algunos errores y ruido en los alineamientos. Estos errores se vuelven más frecuentes a medida que se incluyen secuencias más divergentes en un MSA, y pueden afectar seriamente los análisis subsecuentes. A veces, una secuencia divergente se incluye en un grupo de secuencias conservadas o cercanas (en términos evolutivos). Esta inclusión puede alterar todo el alineamiento, y en ese caso, se deben introducir brechas para mantener áreas homólogas y conservadas. En muchos casos, las brechas se insertan erróneamente, y no solo se vuelven no informativas, sino que también pueden disminuir la calidad del alineamiento global. La distorsión introducida en el alineamiento afecta no solo a algunas columnas, sino a casi todo el MSA [22].

En la bioinformática, la medida práctica de la calidad de un alineamiento candidato para reflejar con precisión las relaciones de homología y los eventos evolutivos se llama función objetivo. La función objetivo genera una puntuación numérica para cada posible solución se buscar optimizar ese valor para hallar la que tiene la mejor puntuación.

Dos medidas populares para puntuar alineamientos múltiples completos son la puntuación de suma de pares (SP) y la puntuación de columna (CS). Sin embargo, estas puntuaciones solo se pueden utilizar si se dispone de un alineamiento de referencia de las mismas secuencias. La puntuación SP calcula la proporción de pares de residuos alineados de manera idéntica en los alineamientos de prueba y de referencia, mientras que la puntuación CS mide la fracción de posiciones alineadas de manera idéntica [23].

1.4.3. Matrices para calcular la similitud en un MSA

Si bien, un alineamiento entre pares genera alineamientos cercanos, es importante también incluir información evolutiva al momento de buscar las relaciones entre las secuencias. Una manera de lograr esto es incluir cuantos cambios de bases (en el caso de ADN/RNA) o amino ácidos (en el caso de proteínas) ocurrieron a través del tiempo. Para ello, se empezaron a utilizar matrices de similitud para asignar un puntaje a dos valores que no coinciden al momento de compararlos de a pares [24].

PAM: Point accepted Mutation

En la década de 1970, la investigadora Margaret Dayhoff junto a sus colaboradores investigaron los cambios evolutivos en las cadenas de aminoácidos de las proteínas que habían recopilado en su libro llamado *Atlas of Protein Sequence and Structure* de 1978[25]. Estudiaron más de 1500 cambios entre más de 70 grupos de proteínas donde cada uno compartía mas del 85 por ciento de identidad. Como resultado de este estudio, observaron los cambios más frecuentes y los que nunca se pueden dar, dando lugar al valor conocido como *mutación puntual aceptada* (*PAM* por *Point accepted mutation* en inglés) y normalizando estos valores, confeccionaron la matriz *PAM1*, cuantificando la probabilidad de que un aminoácido sea reemplazado por otro en un intervalo evolutivo de 1 *PAM*, donde cada intervalo evolutivo *PAM* se define como el cambio del uno por ciento de los residuos en el alineamiento de dos secuencias, es decir, un cambio cada 100 residuos.

En base a los valores *PAM* se puede obtener una métrica llamada *odd-ratio* que cuantifica la probabilidad de que una sustitución se de en una posición dada, por ejemplo un *odd-ratio* de diez implica que la sustitución es diez veces más frecuente que la probabilidad de encontrar alineados ambos aminoácidos. En contraposición, un *odd-ratio* de 0.5 significa que la probabilidad de encontrar alineados ambos aminoácidos tiene el doble de probabilidad que la mutación. Las matrices de *odd-ratios*, son la base de las matrices denominadas *PAM* para calcular la similitud en un alineamiento múltiple de secuencias.

BLOSUM: BLOck of Amino Acid SUbstitution Matrix

Con el tiempo surgieron nuevas matrices con mejor precisión, entre ellas las matrices *BLOSUM* (sigla de la expresión en inglés *BLOck of Amino Acid SUbstitution Matrix*) publicadas

en 1992 por S. Henikoff y J. Henikoff [26]. En este caso, la definición de las matrices se hizo en base a 500 grupos de secuencias con el fin de mejorar alineamientos de secuencias divergentes. La matriz **BLOSUM62** usó proteínas con una identidad igual o mayor a 62 por ciento. Las matrices de tipo *PAM* fueron generadas mediante extrapolación de datos de alineamientos de secuencias muy conservadas, mientras que las matrices *BLOSUM* son creadas a partir de secuencias reales menos conservadas [27].

Gonnet92

La matriz de sustitución Gonnet92 [28], está basada en un análisis exhaustivo de las sustituciones entre secuencias conocidas de proteínas. Este método innovador se distingue por su capacidad de auto-ajustarse a partir de una gran base de datos de secuencias, lo que permite una evaluación muy precisa y adaptable de las similitudes entre proteínas. La metodología empleada por Gonnet92 consiste en comparar secuencias de la base de datos *PIR* y *Swiss-Prot*, ajustando los valores de la matriz según la frecuencia observada de las sustituciones entre aminoácidos en estas secuencias.

1.5. Clasificación de los problemas desde un enfoque informático

Las computadoras solo pueden resolver problemas a través de procesos algorítmicos, o sencillamente conocidos como algoritmos. Por tanto, conocer y evaluar la eficiencia de los algoritmos está fuertemente relacionado con evaluar la complejidad de los mismos. La computabilidad es asociada a encontrar estructuras de forma tal que un *software* escrito en cualquier lenguaje de programación pueda ser ejecutado y resolver un problema [29].

No todos los problemas pueden ser resueltos por métodos computacionales, es por eso que se desarrolló la *Teoría de la Complejidad Computacional* [29] para poder estudiar los recursos requeridos al momento de resolver un problema en forma computacional, dichos recursos usualmente son el tiempo de procesamiento y la memoria utilizada al momento de almacenar las estructuras de datos. Se dice que un cálculo es complejo si es difícil de realizar, siendo uno que requiere más recursos que un problema sencillo. Si un problema necesita más tiempo que otro sencillo, se trata de un problema de complejidad temporal, mientras que si se trata de espacio de almacenamiento, será un problema de complejidad espacial. Sobra decir que los tiempos de procesamiento son cada vez menores, dado que los tiempos actuales son muchísimo más breves que hace una década y así será para las generaciones siguientes. Sin embargo, hay pro-

blemas cuyo tiempo de procesamiento puede ser absurdamente largo e inaceptable al momento de obtener una solución.

La máquina de Turing [30] es un modelo extremadamente simple que permite simular cualquier procedimiento computacional y es utilizado al estudiar la complejidad computacional de los problemas. Su importancia radica en que permite clasificar cuáles problemas pueden ser resueltos computacionalmente y cuáles no: un problema es computable si existe un algoritmo que puede ser implementado en una máquina de Turing que resuelve el problema para todas las posibles entradas. Al contrario, no es considerado computable cuando no existe algoritmo que pueda resolverlo para todas las entradas posibles. Esta clasificación es clave para entender los límites de la computación y desarrollar algoritmos eficientes que resuelvan problemas complejos. La teoría *Church-Turing* dice que si una máquina de Turing no puede resolver un problema, entonces ninguna computadora podrá hacerlo debido a que no existe algoritmo que halle una solución [31].

En consecuencia del estudio de la complejidad computacional, se clasificaron los problemas en dos clases. Los que se pueden resolver y los que no tienen solución. Los que tienen solución requerirán de recursos temporales y espaciales cuyo análisis y determinación es fundamental para evaluar su desempeño y factibilidad.

Los problemas que se resuelven en un tiempo relacionado linealmente a su tamaño, son los llamados problemas con un orden de complejidad lineal y las computadoras actuales pueden resolver problemas con complejidades polinómicas. Siendo su clase P, mientras que los problemas con tiempos de resolución no polinomial están bajo la clase NP, es decir, no pueden ser resueltos en un tiempo razonable.

Dentro de este campo de la computación, se creó una clasificación de acuerdo la computabilidad de un problema:

- **Clase L**: Son los problemas que pueden ser resueltos por una Máquina de Turing determinista, utilizando una cantidad de recursos logarítmicamente proporcional al tamaño del problema. En caso de existir una solución, ella es única.

- **Clase P**: Son problemas que se pueden resolver con una Máquina de Turing determinista en un tiempo polinomial o polinómico. Este tipo de problemas pueden resolverse en un tiempo aceptable.

- **Clase NP**: Engloba al conjunto de problemas que no pueden ser resueltos por una una Máquina de Turing determinista en un tiempo polinómico. Sin embargo, las soluciones de

estos problemas tiene demostraciones verificables por cálculos deterministas en tiempos polinómicos.

Los algoritmos exactos (programación dinámica, familia de algoritmos *branch and bound* y A*, método del gradiente descendente, métodos de Newton, entre otros) encuentran la solución óptima de un problema NP, pero pueden necesitar un tiempo tan largo de procesamiento que no es aceptable; por lo que esta consideración hace que otros tipos de algoritmos sean utilizados, denominados aproximados. Estos últimos algoritmos encuentran soluciones óptimas o cercanas al óptimo en un tiempo de procesamiento aceptable. Estos algoritmos incluyen a las heurísticas y metaheurísticas.

2. Enfoques bioinformáticos para realizar un MSA

Construir un MSA no es una tarea computacionalmente trivial dado que implica realizar cálculos complejos, y los resultados no son siempre biológicamente exactos. A su vez, los tiempos de ejecución y uso de recursos computacionales dependen en gran medida de la cantidad de secuencias a alinear, a mayor cantidad, mayor complejidad. Es un problema de optimización perteneciente la clase NP dentro de la Teoría de la Computabilidad. Si bien, con cada innovación tecnológica se mejora la velocidad de ejecución y la calidad los resultados, sigue siendo un problema a optimizar desde distintos enfoques estocásticos.

Suponiendo que se quieren alinear tres o más secuencias, dependiendo la cantidad de espacios, inserciones o deleciones que se incorporen en el resultado, obtendremos una cantidad de posiciones donde los valores coinciden en ambas secuencias. A mayor número de coincidencias, mejor será el alineamiento. Por eso, al momento de hacer este tipo de análisis, se debe considerar que las secuencias sean homólogas o similares, dado que una gran disparidad distorsiona el resultado global del alineamiento múltiple.

En este capítulo, primero, se presentan las formas de alineamiento, conjuntamente con conceptos algorítmicos y probabilísticos que se usan en el estado del arte. Finalmente, se introduce y describe el software para realizar MSA, elegido para realizar la comparación en este trabajo.

2.1. Métodos de MSA

Los métodos de resolución de problemas MSA se pueden clasificar de acuerdo a la longitud de residuos a alinear. Los alineamientos globales son típicamente utilizados para secuencias que se espera que sean similares a lo largo de toda su longitud, mientras que los alineamientos locales y los enfoques híbridos son más apropiados para secuencias que pueden tener regiones de alta similitud intercaladas con regiones de baja similitud.

En el caso de los alineamientos globales, como el algoritmo *Needleman-Wunsch*, la principal ventaja es que producen alineamientos que abarcan toda la longitud de las secuencias. Esto permite capturar las similitudes y diferencias a lo largo de todo el conjunto de datos y facilita la comparación y la identificación de patrones conservados. Sin embargo, una desventaja de los alineamientos globales es que pueden ignorar regiones cortas pero altamente conservadas debido a la penalización de las brechas.

Por otro lado, los alineamientos locales, como el algoritmo *Smith-Waterman*, se centran en identificar regiones de alta similitud en lugar de buscar alineamientos globales. Esto los ha-

ce especialmente útiles para identificar dominios funcionales, motivos conservados o regiones estructurales importantes en secuencias biológicas. Sin embargo, una desventaja de los alineamientos locales es que pueden pasar por alto las similitudes entre regiones no contiguas de las secuencias y no proporcionan un contexto global completo.

Los algoritmos híbridos intentan identificar tanto las similitudes globales como las regiones locales significativas en las secuencias. Proporcionan una visión más completa y detallada de las relaciones entre las secuencias y son útiles cuando se necesita un equilibrio entre la cobertura global y la captura de regiones altamente conservadas. Sin embargo, una posible desventaja de los alineamientos híbridos es que su complejidad computacional puede ser mayor en comparación con los algoritmos globales o locales tradicionales.

2.1.1. Algoritmo Needleman-Wunsch

Este algoritmo fue propuesto por primera vez por Needleman y Wunsch en 1970 [32], utiliza programación dinámica para encontrar el mejor alineamiento global entre múltiples secuencias. La programación dinámica consiste en dividir el problema original en sub-problemas más sencillos, de forma tal que la solución final está formada por las sub-soluciones más sencillas.

Este algoritmo se basa en la construcción de una matriz de puntuaciones, donde cada casilla representa el mejor alineamiento posible entre dos posiciones de las secuencias. Estas puntuaciones se calculan considerando la similitud o la distancia entre ambas. El algoritmo utiliza una función de puntuación, como *BLOSUM*, para determinar la similitud.

El proceso de alineamiento comienza llenando la matriz de puntuaciones. Luego, se busca un camino óptimo en la matriz para encontrar la mejor puntuación y determinar los residuos que deben ser alineados. El algoritmo continúa iterando hasta que se completa toda la matriz y se obtiene el alineamiento óptimo.

Con el paso de los años, se han realizado numerosas mejoras y variaciones a este algoritmo básico para abordar desafíos adicionales, como la penalización de las brechas en el alineamiento[33].

Algoritmo 1 Algoritmo Needleman-Wunsch

1: Inicializar la matriz de puntuaciones con ceros.
2: **for** cada casilla de la matriz, excepto la primera fila y la primera columna **do**
3: Calcular el puntaje de coincidencia si los caracteres de las secuencias son iguales.
4: Calcular el puntaje de eliminación sumando la puntuación de la casilla superior y la penalización por brecha.
5: Calcular el puntaje de inserción sumando la puntuación de la casilla izquierda y la penalización por brecha.
6: Asignar el puntaje máximo entre los puntajes calculados a la casilla actual.
7: **end for**
8: Recorrer la matriz de puntuaciones desde la esquina inferior derecha hasta la esquina superior izquierda:
9: Si el puntaje proviene de una coincidencia o un empate: agregar los caracteres correspondientes a la secuencia alineada.
10: Si el puntaje proviene de una eliminación: agregar un espacio en la secuencia alineada.
11: Si el puntaje proviene de una inserción: agregar un espacio en la otra secuencia alineada.
12: **return** las secuencias alineadas.

2.1.2. Algoritmo Smith-Waterman

El algoritmo *Smith-Waterman* propuesto por Smith y Waterman en 1981 [34] también utiliza programación dinámica para encontrar las regiones de similitud más significativas entre dos secuencias. Pero, diferencia de *Needleman-Wunsch*, el algoritmo se centra en encontrar subsecuencias locales de alta similitud.

Se utiliza una matriz de puntuaciones para determinar las mejores coincidencias entre las posiciones de las secuencias. La principal diferencia radica en el manejo de las puntuaciones negativas. Mientras que en el algoritmo *Needleman-Wunsch* se permite la aparición de puntuaciones negativas para representar los *gaps*, *Smith-Waterman* establece un umbral de cero, evitando así la penalización de coincidencias locales.

El proceso de alineamiento comienza llenando la matriz de puntuaciones de manera similar al algoritmo *Needleman-Wunsch*. Sin embargo, en lugar de mantener un puntaje global máximo, el algoritmo *Smith-Waterman* realiza un seguimiento de los puntajes máximos locales en cada etapa del cálculo. Esto permite identificar las regiones de alta similitud, conocidas como

Algoritmo 2 Algoritmo Smith-Waterman

1: Inicializar la matriz de puntuaciones con ceros.
2: **for** cada casilla de la matriz, excepto la primera fila y la primera columna **do**
3: Calcular el puntaje de coincidencia si los caracteres de las secuencias son iguales.
4: Calcular el puntaje de eliminación sumando la puntuación de la casilla superior y la penalización por brecha.
5: Calcular el puntaje de inserción sumando la puntuación de la casilla izquierda y la penalización por brecha.
6: Asignar el puntaje máximo entre los puntajes calculados a la casilla actual.
7: Si el puntaje máximo es negativo, establecerlo en cero.
8: **end for**
9: Encontrar el puntaje máximo en toda la matriz y ubicar la casilla correspondiente.
10: Recorrer la matriz de puntuaciones desde la casilla con el puntaje máximo hasta llegar a una casilla con puntaje cero:
11: Si el puntaje proviene de una coincidencia o un empate: agregar los caracteres correspondientes a la secuencia alineada.
12: Si el puntaje proviene de una eliminación: agregar un espacio en la secuencia alineada.
13: Si el puntaje proviene de una inserción: agregar un espacio en la otra secuencia alineada.
14: Mover a la casilla vecina con el puntaje máximo.
15: **return** las secuencias alineadas.

alineamientos locales, que pueden ser fundamentales en el análisis de secuencias biológicas.

El algoritmo *Smith-Waterman* ha demostrado ser una buena herramienta para la identificación de regiones de similitud local en secuencias biológicas. Desde su introducción, se han desarrollado variantes y mejoras para adaptarlo a diferentes contextos y necesidades. Por ejemplo, se han propuesto algoritmos acelerados basados en estrategias de filtrado y enfoques heurísticos para manejar conjuntos de datos más grandes y complejos [35].

2.1.3. Complejidad informática al resolver el problema de MSA

Al hablar de la resolución informática de un MSA, es necesario hablar del espacio de búsqueda de las soluciones para un conjunto de secuencias. En este caso, el tamaño de dicho espacio crece enormemente en base a la cantidad de secuencias a alinear y sus longitudes, por lo que el problema se considera NP-Duro (*NP-Hard*).

En un planteo ideal del problema, el uso de la información evolutiva y estructural que existen entre secuencias de una misma familia deberían ser el factor clave para obtener un alineamiento múltiple preciso y realista, sin embargo a veces dicha información no está disponible o es difícil de usar [19].

En la práctica los métodos heurísticos se han ganado su lugar al resolver los alineamientos para todos los conjuntos de secuencias excepto los de corta longitud. Estas heurísticas se basan

comúnmente en estrategias progresivas: primero se construye un boceto de árbol filogenético a partir de las secuencias y gradualmente se arma el alineamiento siguiendo el orden del árbol. El problema se da cuando se comenten errores en los primeros alineamientos dado que no podrán ser rectificados al momento de incorporar nuevas secuencias.

La alternativa principal a los métodos progresivos es un alineamiento simultaneo de todas las secuencias, con el problema de que son muy demandantes a nivel computacional dado que requieren hacer un uso intensivo de procesamiento y memoria. Si bien no garantizan que encuentren la solución óptima, son suficientemente robustos y menos sensibles al número de secuencias que los métodos deterministas. En general estos métodos intentan conseguir un alineamiento global, pero también hay algunos métodos que intentan hacer alineamientos locales, y son de gran utilidad si las secuencias comparten un solo dominio. Por lo que un método que pueda combinar lo mejor de los alineamientos globales y locales sería realmente un alineador múltiple de secuencias poderoso, flexible y preciso.

Hay propuestas que se basan en Modelos Ocultos de Markov (abreviado *HMM* por sus siglas en inglés) [36]. Con el fin de resolver el problema de MSA, también se han creado métodos que buscan optimizar una función objetivo, mediantes métodos metaheurísticos. Entre ellos algoritmos genéticos como SAGA [37], o incluso SA [38].

2.1.4. Acerca de la fusión progresiva

Los enfoques de fusión progresiva permiten combinar de manera eficiente los alineamientos parciales obtenidos en los pasos anteriores y generar un alineamiento múltiple final que captura las relaciones entre todas las secuencias de entrada.

Un método comúnmente utilizado en el enfoque de fusión progresiva es el algoritmo de progresión múltiple (*M-CLUSTAL*), que fue propuesto por Higgins y Sharp en 1988 [39]. Este algoritmo utiliza una técnica de agrupamiento jerárquico para fusionar los alineamientos parciales. Comienza agrupando las secuencias más similares en subconjuntos y realiza alineamientos progresivos entre estos subconjuntos utilizando el algoritmo de alineamiento múltiple, como el algoritmo de Clustal. Luego, los subconjuntos se fusionan en un alineamiento múltiple completo.

Otro enfoque popular es el método de progresión progresiva introducido por Feng y Doolittle en 1987 [40]. Este método también utiliza una estrategia de agrupamiento jerárquico, pero utiliza distancias entre secuencias para construir un árbol guía y fusionar gradualmente los alineamientos parciales. Las secuencias se agrupan en función de las distancias genéticas y se

realiza un alineamiento progresivo a medida que se fusionan los grupos.

2.1.5. Acerca de los Modelos Ocultos de Markov (HMM)

Es un enfoque probabilístico, introducido por Baum y sus colaboradores [41] que ha demostrado ser una herramienta poderosa en la biología computacional y en una variedad de campos. En la biología computacional, los HMM se han utilizado para el análisis de secuencias de ADN y ARN, permitiendo la identificación de genes, el alineamiento de secuencias y la predicción de estructuras secundarias [42]. La idea es que una secuencia observable es el resultado de un proceso estocástico subyacente no observable. Este proceso latente se modela mediante una cadena de Markov con estados ocultos, mientras que las observaciones se generan a partir de cada estado oculto mediante distribuciones de probabilidad condicional. Esta representación permite modelar la dependencia temporal de las secuencias y capturar la estructura subyacente.

Un HMM se caracteriza por su conjunto de estados ocultos, las probabilidades de transición entre estos estados, las distribuciones de probabilidad condicional asociadas a cada estado y las probabilidades iniciales de los estados. Estos parámetros se estiman a partir de datos de entrenamiento utilizando algoritmos como el algoritmo de Baum-Welch, que emplea el algoritmo de avance y retroceso para calcular las estimaciones máximas.

Acerca de su funcionamiento, el algoritmo del HMM (algoritmo 3) utiliza las matrices α y β para calcular la probabilidad de la secuencia observada y encontrar la secuencia de estados ocultos más probable. Esto se logra mediante el uso de probabilidades de transición, distribuciones de probabilidad condicional y probabilidades iniciales, que se combinan con las observaciones para estimar la secuencia oculta:

1. **Inicialización**: Se inicializan dos matrices, llamadas alfa (α) y beta (β), con ceros. Estas matrices se utilizan para almacenar valores intermedios durante el cálculo.

2. **Paso de avance**: Se calculan los valores de la matriz alfa utilizando el algoritmo de avance. Para cada tiempo t y cada estado j, se calcula el valor de α_t(j) que representa la probabilidad de estar en el estado j en el tiempo t dado las observaciones anteriores.

3. **Paso de retroceso**: Se calculan los valores de la matriz beta utilizando el algoritmo de retroceso. Para cada tiempo t y cada estado i, se calcula el valor de β_t(i) que representa la probabilidad de estar en el estado i en el tiempo t dado las observaciones futuras.

4. **Cálculo de la probabilidad de la secuencia observada**: Se calcula la probabilidad total de la secuencia observada sumando los valores de alfa en el último tiempo T.

5. **Cálculo de la secuencia de estados ocultos más probable**: Utilizando los valores de alfa, beta y la probabilidad de la secuencia observada, se determina la secuencia de estados ocultos más probable. Para cada tiempo t, se selecciona el estado i que maximiza el producto de α_t(i) y β_t(i) dividido por la probabilidad de la secuencia observada.

6. **Retorno**: Se devuelve la secuencia de estados ocultos más probable.

Algoritmo 3 Algoritmo del HMM

Require: Secuencia de observaciones $O = O_1, O_2, \ldots, O_T$

Require: Conjunto de estados ocultos $S = \{S_1, S_2, \ldots, S_N\}$

Require: Probabilidades de transición $A = \{a_{ij}\}$, donde a_{ij} es la probabilidad de transición de S_i a S_j

Require: Distribuciones de probabilidad condicional $B = \{b_j(O_t)\}$, donde $b_j(O_t)$ es la probabilidad de observar O_t dado el estado S_j

Require: Probabilidades iniciales $\pi = \{\pi_i\}$, donde π_i es la probabilidad inicial del estado S_i

Ensure: La secuencia de estados ocultos más probable $Q = q_1, q_2, \ldots, q_T$

Inicializar α y β con ceros
Calcular α utilizando el algoritmo de avance:
    $\alpha_1(i) = \pi_i \cdot b_i(O_1)$
    $\alpha_t(j) = \left(\sum_{i=1}^{N} \alpha_{t-1}(i) \cdot a_{ij}\right) \cdot b_j(O_t)$ para $t = 2$ a T y $j = 1$ a N
Calcular β utilizando el algoritmo de retroceso:
    $\beta_T(i) = 1$ para $i = 1$ a N
    $\beta_t(i) = \sum_{j=1}^{N} a_{ij} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j)$ para $t = T-1$ a 1 y $i = 1$ a N
Calcular la probabilidad de la secuencia observada:
    $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$
Calcular la secuencia de estados ocultos más probable Q utilizando α y β:
    $q_t = \arg\max_i \left(\frac{\alpha_t(i) \cdot \beta_t(i)}{P(O|\lambda)}\right)$ para $t = 1$ a T
return Q

2.2. Software de la literatura elegido para MSA

Si bien la cantidad de software para realizar alineamiento múltiple es grande y diversa, para realizar la comparación propuesta en este trabajo se han escogido solo cinco de los más utilizados por la comunidad científica.

- **Clustal Omega**: Es muy eficiente en términos de tiempo y memoria, lo que lo hace adecuado para alinear grandes conjuntos de datos de secuencias. Clustal Omega también tiene opciones para ajustar los parámetros del alineamiento, lo que permite a los usuarios adaptar el proceso a sus necesidades específicas [43].

- **MUSCLE** (MUltiple Sequence Comparison by Log-Expectation): es otro algoritmo muy utilizado que a menudo se considera más preciso que Clustal Omega, especialmente para secuencias más divergentes. Sin embargo, puede ser más lento y consumir más memoria que Clustal Omega para grandes conjuntos de datos [44].

- **KAlign**: es conocido por su velocidad y eficiencia, lo que lo hace adecuado para alinear grandes conjuntos de datos. Sin embargo, la precisión del alineamiento puede no ser tan alta como la de MUSCLE o Clustal Omega para algunos conjuntos de datos [45].

- **T-Coffee** (Tree-based Consistency Objective Function For alignmEnt Evaluation): es un algoritmo único que combina información de varios métodos y fuentes para mejorar la precisión del alineamiento. Aunque puede ser más lento que otros métodos, a menudo produce alineamientos de alta calidad.

- **MAFFT** (Multiple Alignment using Fast Fourier Transform): es un programa de alineamiento muy versátil que incluye varias opciones y estrategias para diferentes tipos de datos de secuencias. MAFFT es conocido por su equilibrio entre velocidad y precisión [46].

En la tabla 3 se realiza una comparación de diferentes programas de software MSA utilizados en el estado del arte clasificando estos programas de *software* de acuerdo a lo explicado anteriormente: algunos implementan un enfoque global, alineando las secuencias completas a la vez, mientras que otros adoptan un enfoque híbrido, combinando alineamientos globales y locales.

Programa	Clasificación de algoritmo	Descripción
Clustal Omega	Global	Algoritmo basado en el algoritmo Needleman-Wunsch, que es un algoritmo global. Usa una estrategia de alineamiento progresivo para alinear todas las secuencias a la vez.
M.U.S.C.L.E.	Global	Utiliza una estrategia de alineamiento progresivo, similar a Clustal Omega. En primer lugar, realiza un alineamiento rápido para agrupar las secuencias y, a continuación, realiza un alineamiento más lento y preciso.
KAlign	Global	También es un programa de alineamiento de secuencias global que utiliza una estrategia de alineamiento progresivo.
T-Coffee	Híbrido (glocal)	Algoritmo híbrido que combina la estrategia de alineamiento progresivo con la incorporación de información de alineamientos locales para mejorar la precisión [19].
MAFFT	Global/Híbrido (glocal)	Puede utilizarse tanto para alineamientos globales como para alineamientos híbridos. Utiliza una estrategia de alineamiento progresivo, pero también puede incorporar información de alineamientos locales [46].

Tabla 3: Comparación de programas de alineamiento de secuencias

Clustal Omega, MUSCLE, KAlign, T-Coffee y MAFFT, son utilizados ampliamente en la Bioinformática. Según Google Scholar en mayo de 2023:

- El artículo original de Clustal Omega, tiene más de 13.756 citas.
- El artículo de MUSCLE ha sido citado 8.545 veces.
- El artículo de T-Coffee tiene 8.295 citas.
- El artículo de MAFFT ha sido citado 13.199 veces.
- El artículo de KAlign ha sido citado 820 veces.

3. Implementación de SA para el problema MSA

En este capítulo se desarrolla una nueva propuesta algorítmica, denominada MSASA, para realizar alineamientos de múltiples secuencias. Esto requiere que antes se expliquen los conceptos de algoritmos heurísticos y metaheurísticos para resolver problemas del tipo de los MSA. Pues, los métodos heurísticos proveen a este trabajo el marco de trabajo para la resolución de variados problemas y es adaptado para realizar alineamientos múltiples de secuencias.

3.1. Heurísticas y metaheurísticas

En oposición a los métodos algorítmicos exactos, las heurísticas son métodos eficientes que permiten encontrar soluciones suficientemente buenas en tiempos considerablemente menores que los métodos exactos. En este ámbito, lo heurístico se usa en contraposición a lo exacto [47].

La optimización se tratar de hallar la mejor solución posible para un determinado problema cuando existen diferentes soluciones y un criterio de elección de una entre todas ellas. Estos tipos de problemas se pueden definir como la búsqueda del valor de las variables de decisión para los que una determinada función objetivo alcance su valor máximo o mínimo. Mientras que el valor de las variables de decisión suelen estar sujetas a restricciones propias del problema a resolver. Hay tipos de problemas de optimización que suelen ser fáciles de resolver. Por ejemplo, el método *Simplex* puede resolver problemas lineales donde la función objetivo y las restricciones son expresiones lineales. Sin embargo, existen problemas complejos que no pueden ser resueltos mediante un procedimiento de búsqueda exhaustivo o completo, ya sea porque tal método no existe o su complejidad crece de manera no polinomial [48].

Dentro de la optimización, aparecieron procedimientos conocidos como métodos heurísticos [49] para hallar soluciones aproximadas donde la velocidad del procedimiento para encontrar una solución es igual de importante que la calidad de la misma. Estos métodos dependen en gran medida del problema para el que se han diseñado, entonces no se puede asegurar que existe una heurística capacitada en obtener los mejores resultados de cualquier problema de optimización [50].

Los métodos metaheurísticos [51] aparecieron para encontrar mejores soluciones que las halladas por los modelos heurísticos tradicionales. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos [52]. Las metaheurísticas son las que guían el diseño de las heurísticas dando un marco de trabajo para diseñar

un método heurístico que resuelva aproximadamente un problema.

Estos métodos pueden aplicase cuando el problema cuenta con algunas de las siguientes características[53]:

- No se conoce un método exacto que resuelva el problema.
- Si existe un método exacto, su uso es extremadamente costoso a nivel computacional y lo hace inviable.
- Las condiciones que definen la solución son difíciles de implementar con un método exacto.
- El modelo es muy complejo desde el punto de vista lógico.
- Al asumir aproximaciones en pos de simplificar el problema ocasionan que las estructuras del modelo vitales para modelar el contexto del mundo real sean destruidas.
- El uso de un método heurístico es parte de un proceso global que garantiza encontrar la solución óptima, dado que: o bien el método heurístico devuelve una buena solución inicial, o es un paso intermedio del procedimiento, por ejemplo para seleccionar el valor de entrada del proceso siguiente.

Como se menciona anteriormente, la solución encontrada puede no ser la óptima, y esto obliga a evaluar la eficiencia del método:

- El método debe ser **eficiente** en términos de uso computacional
- La solución debe ser **buena** y encontrarse cerca de la solución óptima
- El algoritmo debe ser **robusto** para que una solución mala, es decir alejada del valor óptimo, tenga una baja probabilidad de ser elegida.

Es fundamental que una metaheurística sea evaluada tanto a partir de su eficiencia como de su eficacia. Lo primero se puede contrastar experimentalmente al evaluar los tiempos computacionales necesarios para resolver con éxito los problemas considerados, siendo satisfactorio que los tiempos sean moderados o al menos razonables. Mientras que, la eficacia de una metaheurística debe ser validada en base a un conjunto de casos de prueba con soluciones conocidas. En caso de no tener este banco de pruebas, se debe construir uno recurriendo a otros procesos o técnicas.

En virtud de la naturaleza de las heurísticas y metaheurísticas se proponen varias maneras de medir la calidad de una solución:

- **Comparación con la solución óptima**: Puede que se haya obtenido un valor óptimo de un subconjunto de tamaño reducido dentro del espacio de búsqueda, y dicho valor puede ser de ayuda para compararlo luego contra las soluciones obtenidas por la heurística. La idea es medir el desvío entre la solución hallada y la de referencia, intentando luego ir reduciendo esa desviación.

- **Comparación con una cota**: Cuando el valor óptimo no puede ser conocido ni siquiera con un subconjunto limitado, se puede utilizar una cota inferior si el objetivo es minimizar o superior si es un problema de maximización. Mientras más cerca este la cota al valor óptimo, mejore serán los resultados.

- **Comparación con un método exacto truncado**: Se puede utilizar un algoritmo exacto para obtener un valor de referencia siempre y cuando se pueda limitar el tiempo de ejecución del mismo, ya sea por un limite de iteraciones o de tiempo.

- **Comparación con otras heurísticas**: En problemas difíciles (NP Duros) se suele comparar los resultados con otros métodos empleados previamente de los cuales se conocen que sus soluciones son buenas.

- **Análisis del peor caso**: Fue utilizado en el pasado pero sus resultados no suelen ser buenos durante el comportamiento medio del algoritmo, además de ser complicado de implementar por métodos heurísticos muy sofisticados. Su metodología consiste en tomar a los ejemplos más desfavorables para el algoritmo y acotar la máxima desviación respeto al óptimo del problema a resolver, logrando de esta forma, acotar el resultado para cualquier valor, lo que representa su gran ventaja.

3.1.1. Clasificación de las metaheurísticas

Dentro del universo de metaheurísticas posibles, se pueden mencionar los siguientes grupos de métodos:

- **Metaheurísticas inspiradas en la física:** Este campo de la ciencia es fuente de inspiración de diversos métodos, entre ellos la metaheurística de enfriamiento simulado, SA, que se basa en modelar el comportamiento de un cuerpo sólido al ser calentado y posteriormente enfriado para obtener estados de baja energía [54].

- **Metaheurísticas inspiradas en la Teoría de la Evolución:** Donde estos métodos van construyendo un grupo de soluciones, en lugar de los otros que van pasando una solución de iteración a iteración. El concepto general implica generar individuos que representan posibles soluciones, seleccionar algunos, combinarlos y reemplazar la generación anterior con nuevas soluciones mejores que sus antecesores. Un ejemplo de este tipo de metaheurística son los algoritmos genéticos publicados por el investigador John Holland de la Universidad de Michigan a finales de la década de 1960 quien se baso en los mecanismos de selección natural que enuncia que los individuos mas aptos sobreviven al adaptarse más fácilmente a los cambios en el ambiente [55].

- **Metaheurísticas basadas en la biología:** Son aquellos métodos que toman conceptos de la naturaleza tales como la inteligencia colectiva, por ejemplo optimizaciones basadas en colonias de hormigas o de abejas [56]. En el caso de las hormigas, el método *Ant Colony Optimization* simula computacionalmente la comunicación indirecta de los individuos mediante una sustancia química conocida como feromonas para establecer el camino más corto de un punto a otro [48].

3.2. La metaheurística SA

El propósito principal de SA [5] es buscar el mínimo global de una función objetivo, en lugar de quedar atrapado en un mínimo local como otros algoritmos de optimización. Este método metaheurístico está inspirado en el enfriamiento o recocido de una pieza sólida de metal donde se establece una analogía entre el proceso termodinámico de enfriamiento y la búsqueda de una solución en forma heurística [57]:

- **Se calienta un solido metálico cristalino**: El algoritmo comienza con un estado inicial y una temperatura alta.

- **Se enfría muy lentamente dicha pieza solida**: En cada iteración, el algoritmo selecciona un estado vecino de manera aleatoria y calcula la diferencia de energía (la diferencia en la función objetivo) entre el estado vecino y el estado actual. Si la energía del estado vecino es menor que la del estado actual (es decir, la función objetivo es menor), el estado vecino se convierte en el nuevo estado actual.

- **La pieza está fría**: Se alcanza un estado de mínima energía con una estructura cristalina mucho más regular y ordenada que al inicio del proceso de enfriamiento.

Dentro del vocabulario de esta metodología, se asocian conceptos de la física con el contexto del problema a resolver:

- Un estado físico se asocia a una solución al problema a resolver.
- La energía del estado físico corresponde al costo de la solución.
- La temperatura es el parámetro de control.

Conforme el proceso avanza, la temperatura disminuye según un esquema de enfriamiento predefinido, lo que reduce la probabilidad de aceptar estados peores. Finalmente, el algoritmo converge hacia el mínimo global de la función objetivo. La inclusión de los criterios de Boltzmann y la condición de Metrópolis en SA le otorga una gran robustez frente a problemas de optimización complejos con numerosos mínimos locales (Van Laarhoven y Aarts, 1987) (figura 5).

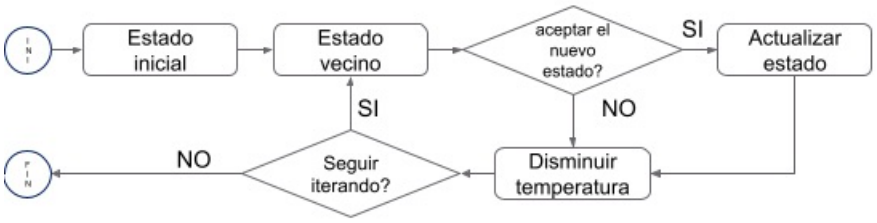

Figura 5: Para el problema de encontrar un MSA, un estado del sistema representa un alineamiento múltiple. Para evaluar el nuevo estado se utiliza un cálculo de la energía, que en este caso, es una función que determina heurísticamente la calidad del MSA generado. Cada nuevo estado vecino, es un MSA con una pequeña modificación.

3.2.1. Criterio de Boltzmann

La importancia del criterio de Boltzmann radica en que permite la posibilidad de aceptar soluciones peores (con una energía más alta) en las etapas iniciales del algoritmo [58]. Si la energía del estado vecino es mayor que la del estado actual, se acepta como nuevo estado actual con una probabilidad que depende de la diferencia de energía y de la temperatura actual, según la fórmula de Boltzmann para la termodinámica estadística. Esta característica permite a SA intentar escapar de potenciales mínimos/máximos locales.

$$Probabalidad_{Boltzman} = e^{(\frac{f(i)-f(j)}{T_i})} \tag{1}$$

En donde el numerador de la fracción corresponde a la diferencia de energías entre el nuevo estado, j y el mejor hasta el momento, i, y el denominador es en función de la temperatura actual del sistema, T_i.

Se puede demostrar que si el parámetro de la temperatura es disminuido lentamente y si se generan suficientes transiciones por cada temperatura analizada, el estado de configuración óptimo es alcanzado. Siendo una transición un proceso de dos etapas: la generación de estados y el mecanismo de aceptación de las mismas.

3.2.2. La condición de Metrópolis

Por su parte, la condición de Metrópolis se utiliza para determinar si se acepta o se rechaza el nuevo estado. Según este criterio, el estado vecino se acepta siempre si su energía es menor que la del estado actual, o si la probabilidad calculada según el criterio de Boltzmann es mayor que un número aleatorio entre 0 y 1 [59]. De esta forma soluciona el problema de quedarse sobre-explorando un área del espacio de búsqueda y por consiguiente, devolver soluciones que son un mínimo o máximo local.

Algoritmo 4 Función Debe Aceptar

Require: nuevo_puntaje
Require: puntaje_actual
Require: temperatura_actual
1: **if** nuevo_puntaje $\geq$ puntaje_actual **then**
2: **return** Verdadero
3: **else**
4: delta $\leftarrow$ nuevo_puntaje - puntaje_actual
5: probabilidad $\leftarrow$ exp(delta / temperatura_actual)
6: **return** aleatorio(0, 1) $<$ probabilidad
7: **end if**

Donde `exp` es la función que devuelve el número E elevado a la potencia pasada por argumento.

3.2.3. Algoritmo general

El proceso iterativo principal comienza desde una temperatura alta que va a ir disminuyendo muy lentamente hasta llegar a una temperatura final. Mientras que el proceso iterativo interno, busca soluciones candidatas en el vecindario de la solución actual. En estados de alta temperatura, es mucho más probable que se acepten soluciones malas, al contrario de los estados con

temperaturas cercanas a la temperatura final donde es probable que las soluciones vecinas no mejoren la marcada como mejor.

La metodología consiste en generar soluciones al azar alrededor de la mejor encontrada hasta el momento y compararlas al medir la energía de ambas. En caso que la energía de la nueva solución generada aleatoriamente sea menor que la de la mejor solución encontrada hasta ese momento, la nueva toma el lugar de la mejor. En caso contrario, puede que la solución vecina igualmente sea aceptada si cumple con el criterio de Metrópolis.

Con el fin de encontrar una solución mínima o máxima global, la adición de este criterio ayuda a salir de las zonas de mínimos o máximos locales. Una solución vecina es aceptada si su energía es menor que la mejor solución o dependiendo de una probabilidad asociada a la temperatura actual del modelo.

Al ejecutar esta condición de aceptación a lo largo de muchas transiciones se puede decir que se alcanza el equilibrio térmico caracterizado por la distribución de Boltzmann (algoritmo 5).

Algoritmo 5 Función Simulated Annealing

Require: secuencias

Require: temperatura_actual

Require: tasa_de_enfriamiento

Require: temperatura_min

Require: limite_sin_cambios

```
1: iteración ← 0
2: sin_cambios ← 0
3: energía_actual ← función_objetivo(secuencias)
4: while temperatura_actual > temperatura_min do
5:     energía_iteración ← energía_actual
6:     nuevo_estado ← generar_nuevo_estado(secuencias)
7:     nueva_energía ← función_objetivo(nuevo_estado)
8:     if debe_aceptar(energía_actual, nueva_energía, temperatura_actual) then
9:         secuencias ← copiar(nuevo_estado)
10:        energía_actual ← nueva_energía
11:    end if
12:    if energía_iteración == energía_actual then
13:        sin_cambios ← sin_cambios + 1
14:    else
15:        sin_cambios ← 0
16:    end if
17:    temperatura_actual ← temperatura_actual × tasa_de_enfriamiento
18:    iteración ← iteración + 1
19:    if sin_cambios ≥ limite_sin_cambios then
20:        break                                  ▷ Interrumpir el ciclo de iteraciones
21:    end if
22: end while
23: return secuencias
```

3.3. Relación entre el dominio de MSA y la metaheurística SA

El proceso de SA en el contexto de SA implicara comenzar la ejecución del algoritmo con un alineamiento inicial de las secuencias (el material metálico sólido) y luego hacer pequeñas

modificaciones (los estados vecinos) para tratar de mejorar la métrica de calidad del alineamiento múltiple (reducir la energía). Al igual que en el enfriamiento simulado real, la temperatura en SA controla la probabilidad de que se acepten cambios que aumenten la energía, lo que permite la exploración de varias soluciones potenciales y ayuda a evitar quedar atrapado en mínimos locales.

- **El material metálico sólido - Alineamiento Múltiple de Secuencias**: Cada MSA puede ser considerado como una *pieza metálica* a enfriar en el proceso de SA. Esta pieza es el estado que estamos tratando de optimizar.
- **Energía de un Estado - Métrica de Calidad del MSA**: La *energía* en SA se puede considerar como una forma de medir la calidad de una solución. En este caso, es la métrica de calidad del MSA. Por lo general, un MSA de alta calidad (es decir, con poca *energía*) tendrá una gran similitud entre las secuencias y un mínimo de *gaps*. Pero para simplificar los cálculos, la implementación SA para generar MSA va a intentar máximizar este valor.
- **Estado Vecino - MSA con una pequeña modificación**: En el SA, un estado vecino es simplemente un estado que es una pequeña modificación del estado actual. En el caso del alineamiento de secuencias, podría ser un alineamiento con una pequeña modificación, como la adición o eliminación de un *gap* en una o más de las secuencias.

3.3.1. Estructura de datos

La estructura de datos básica (tabla 4) para almacenar un estado dentro del modelo de SA es una matriz de *N filas por M columnas*, donde N es la cantidad de secuencias de entrada y M es la longitud de la secuencia con más residuos (o *gaps*).

Secuencia	C_1	C_2	C_3	...	C_m
$SECUENCIA_1$	M	V	L	...	Y
$SECUENCIA_2$	V	C	D	...	Y
$SECUENCIA_3$	Y	F	–	...	–
...	A	C	D	...	Y
$SECUENCIA_n$	A	C	D	...	F

Tabla 4: Esquema de la matriz que representa un estado del sistema.

Es fundamental que todas las filas tengan la misma longitud, por esta razón, las secuencias que tengan una longitud menor a la secuencia más larga, serán completadas con el símbolo de

gap, el cual representa un posible *InDel* (inserción o deleción). Este proceso de normalizar las longitudes se hace al azar en la primera o última columna de la matriz (tabla 5).

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	V	L	Y	C	D	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 5: Ejemplo de una matriz que representa un estado del sistema con símbolos de *gap* completando las longitudes.

3.3.2. Estados del material metálico sólido

En este caso, un estado es un alineamiento múltiple de secuencias completo, es decir, que todas las secuencias son alineadas con todas las demás. Computacionalmente, esto se logra, guardando el alineamiento en una matriz donde cada fila es una secuencia y cada columna corresponde a una posición dentro del alineamiento. La cantidad de columnas está determinada por la longitud de la secuencia más larga, y en caso de tener una longitud menor, se completa la fila con el símbolo de *gap*.

3.3.3. Generación de los estados vecinos

Para la creación de un nuevo estado se parte de un estado anterior (figura 6), es decir, se manipula una copia de matriz que contiene la representación del alineamiento múltiple y se efectúa un cambio: agregar o remover un símbolo de *gap* en una de las secuencias del estado (algoritmo 6). Para ello, primero se determina al azar si la modificación es una inserción o una eliminación con una probabilidades por defecto del 50 %. Si la acción a realizar es una modificación y hay *gaps* a remover, se efectúa una inserción.

Se procede luego a buscar una secuencia al azar para aplicarle la modificación. Una vez establecida la secuencia a modificar se remueve aleatoriamente uno de los *gaps* al azar o se agrega uno nuevo en una posición al azar.

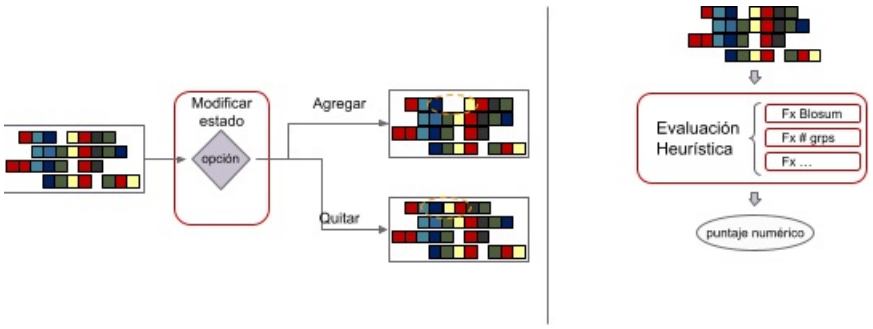

Figura 6: A la izquierda se presenta el esquema del proceso de modificación de un estado para obtener uno nuevo. El mismo puede ser mediante la incorporación o eliminación de un *gap*. Una vez que se decide la modificación a realizar, se toma una secuencia al azar y se le aplica el cambio. A la derecha, el esquema del proceso por el cual se evalúa heurísticamente un estado.

Algoritmo 6 Generación de nuevas secuencias con inserción o eliminación de *gaps*

Require: secuencias
Require: probabilidad_adición_deleción = 0.5 ▷ Misma probabilidad por defecto
1: dirección_completar ← valor aleatorio entre 0 y 1
2: posiciones_gaps ← buscarPosicionesConGap(secuencias)
3: **if** valor aleatorio < probabilidad_adición_deleción **o** posiciones_gaps está vacío **then**
4: secuencias ← agregarGap(secuencias)
5: **else**
6: secuencias ← removerGap(secuencias)
7: **end if**
8: **return** secuencias

En el algoritmo 6 se utilizan las siguientes funciones:

- La función **buscarPosicionesConGap** se encarga de hacer una búsqueda dentro de las posiciones del vector que contengan un símbolo de *gap* y retornar sus índices.
- La función **removerGap** hace una operación de unión entre dos sub-vectores, el primero desde el inicio del vector original hasta la posición del *gap* seleccionado anteriormente, y el segundo desde la posición siguiente a dicho *gap* y el final del vector original.
- La función **agregarGap** consiste en unir tres sub-vectores, el primero va desde la posición inicial del vector original hasta la posición del nuevo *gap*, el segundo es un vector que contiene un solo elemento, el símbolo de *gap*, y el último es un vector que va desde la posición siguiente al *gap* hasta el final del vector original.

A fin de evitar que la matriz crezca con columnas de *gap* tanto en las columnas del principio como las del final, se ejecuta una validación para eliminar el excedente de columnas puramente conformadas por *gaps* como se muestra en el algoritmo .

Algoritmo 7 Eliminación de columnas conteniendo únicamente el símbolo *gap* al inicio y al final de la matriz

Require: secuencias

1: **while** columna(secuencias, 0) contiene_solo *gap* **o** columna(secuencias, -1) contiene_solo *gap* **do**
2: **if** la primera columna de secuencias contiene solo *gap* **then**
3: secuencias ← secuencias desde la segunda columna hasta la última
4: **end if**
5: **if** la última columna de secuencias contiene solo *gap* **then**
6: secuencias ← secuencias desde la primera columna hasta la penúltima
7: **end if**
8: **end while**
9: **return** secuencias

En el algoritmo se utilizan las siguientes funciones:

- La función **columna** devuelve el contenido de una columna dentro de la matriz de secuencias. Para obtener la primera columna se usa el índice 0, mientras que la última se obtiene con -1.
- La función **contiene_solo** valida si todos los elementos de una lista son idénticos al valor pasado por argumento.

Ejemplo de la adición de un símbolo de *gap*

Suponiendo que el estado inicial es el mostrado en la tabla 6, se obtiene al azar la secuencia a modificar, $SECUENCIA_3$, y a continuación se determina en forma aleatoria que la posición a cambiar es C_4 (tabla 7). Entonces, se debe agregar el símbolo de *gap* en la cuarta columna y desplazar el resto del vector una posición hacia adelante como se muestra en la tabla 8. Para finalizar, se verifica que la longitud de todas las secuencias se corresponda con la de de mayor longitud, que en este caso es la SECUENCIA_3, y se completa con *gaps* si es necesario (como se observa en la tabla 10).

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	V	L	Y	C	D	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 6: Ejemplo de una matriz a modificar agregando un símbolo de *gap*.

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	V	L	Y	C	D	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 7: Ejemplo de una matriz a modificar agregando un símbolo de *gap* una vez determinada la secuencia a modificar, en este caso, en la secuencia 3 y la columna 4.

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_10
$SECUENCIA_1$	M	V	L	Y	C	D	–	–	–	
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y	
$SECUENCIA_3$	Y	F	A	–	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–	
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–	

Tabla 8: Ejemplo la matriz modificada en su estado intermedio.

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_10
$SECUENCIA_1$	M	V	L	Y	C	D	–	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y	–
$SECUENCIA_3$	Y	F	A	–	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–	–

Tabla 9: Ejemplo de una matriz modificada con todas las secuencias del mismo largo, completando los espacios vacíos con símbolos de *gap*.

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	V	L	Y	C	D	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	–	A	D	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 10: Una vez finalizada la edición de las secuencias, se procede a remover columnas que solo contengan *gaps* en la primer y última columna.

Ejemplo de la deleción de un símbolo de *gap*

Suponiendo que el estado inicial es el presentado en la tabla 11, se obtiene al azar la secuencia a modificar,$SECUENCIA_1$, y a continuación se determina en forma aleatoria la posición a modificar, C_2, (ver en tabla 12). Luego, se elimina el símbolo de *gap* en la segunda columna y desplaza el resto del vector una posición hacia atrás, como se muestra en la tabla 13. Finalmente, se controla que la longitud de todas las secuencias sea igual a la de mayor longitud, que en este caso es la $SECUENCIA_2$, y rellene con *gaps* si es necesario (ver en (tabla 14).

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	-	L	Y	C	D	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 11: Ejemplo de una matriz a modificar, eliminando un símbolo de *gap*.

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	-	L	Y	C	D	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 12: Ejemplo de una matriz a modificar, eliminando un símbolo de *gap*, en este caso, en la secuencia 1 y la columna 2.

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	L	Y	C	D	–	–	–	
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 13: Ejemplo la matriz modificada en su estado intermedio luego de quitar el símbolo de *gap*.

Secuencia	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
$SECUENCIA_1$	M	L	Y	C	D	–	–	–	–
$SECUENCIA_2$	V	C	D	E	F	Y	V	D	Y
$SECUENCIA_3$	Y	F	A	A	D	–	–	–	–
$SECUENCIA_4$	A	C	D	Y	C	V	–	–	–
$SECUENCIA_5$	A	C	D	F	–	–	–	–	–

Tabla 14: Ejemplo de una matriz modificada con todas las secuencias del mismo largo, completando los espacios vacíos con símbolos de *gap* a la derecha de la matriz. Igualmente, el proceso de completado es definido aleatoriamente entre el comienzo y el final de la matriz.

3.3.4. Energía de un estado

Lo siguiente a tener en cuenta en SA es el cálculo de la energía de un estado, es decir la función objetivo, porque es mediante este valor que el algoritmo decide si un estado vecino va a reemplazar al actual. Para el cálculo de la energía se necesita un método que recibe una representación de un estado y devuelve un valor numérico asociado a la energía de ese estado.

Diversas alternativas heurísticas se implementan en este trabajo para determinar cuál es el mejor resultado MSA al contrastarlo con valores de referencia, pero independientemente de una en particular, el procedimiento es el mismo (algoritmo 8): asociar la energía del sistema a un puntaje que represente la calidad del alineamiento representado en un estado del sistema.

Se itera columna por columna para obtener un puntaje por cada columna de acuerdo a la estrategia elegida y luego se retorna la sumatoria de todos los puntajes dividida por un denominador a elegir por cada estrategia (tabla 15). Dado que el cálculo del puntaje de cada par de columna es independiente de las demás, una posible mejora de rendimiento es la paralelización del cálculo de puntajes, la cual es una de las mencionadas en [60].

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	...	C_m
$SECUENCIA_a$	M	V	Y	M	-	-	M	...	Y
$SECUENCIA_b$	M	I	Y	M	-	M	-	...	I
$SECUENCIA_c$	M	I	Y	I	-	M	-	...	I
$SECUENCIA_d$	M	I	Y	I	-	M	-	...	I
$SECUENCIA_e$	M	I	Y	T	-	M	T	...	I
...	M	...	...	...	...	...	...	...	...
$SECUENCIA_n$	M	V	V	S	-	M	M	...	I
Puntaje	P_1	P_2	P_3	P_4	P_5	P_6	P_7	**...**	P_m

Tabla 15: Pequeño ejemplo de los puntajes obtenidos por columna en una matriz de *n* secuencias y una longitud máxima de residuos de *m*.

Algoritmo 8 Función Objetivo genérica

Require: secuencias

Require: cantidad_secuencias

Require: longitud_máxima

Require: columnas

```
puntaje_total ← 0
for columna en columnas(secuencias) do
    puntaje_columna ← heurística(columna)
    puntaje_total ← puntaje_total + puntaje_columna
end for
return puntaje_total / divisor(cantidad_secuencias, longitud_máxima)
```

Donde:

- La función **columnas** convierte la matrix de secuencias en una lista de listas, donde cada elemento es una columna.
- La función **heurística** recibe una columna y efectúa el cálculo de energía dependiendo de la estrategia elegida.
- La función **divisor** recibe la cantidad de secuencias y la longitud máxima de residuos y devuelve un valor real para dividir el puntaje total. Por defecto, la función devuelve 1,

pero puede modificarse según la heurística, por ejemplo, para dividir la puntación sobre la cantidad de columnas del alineamiento.

$$puntaje\_total = \frac{\sum puntaje\_columna}{divisor(cantidad\_secuencias, longitud\_máxima)}$$

En este trabajo todas las heurísticas implementadas y evaluadas trabajan columna por columna. Hay dos tipos de evaluaciones: las primeras analizan las columnas como un todo (heurísticas *Coincidencias*, *Identidad*) y las otras procesan las combinaciones de residuos de una columna (*Similitud*, *Global* y *Local*). En el último caso, la cantidad de cálculos a realizar por columna crece en forma exponencial (figura 7) y por consecuencia, el tiempo requerido para el cálculo del puntaje total.

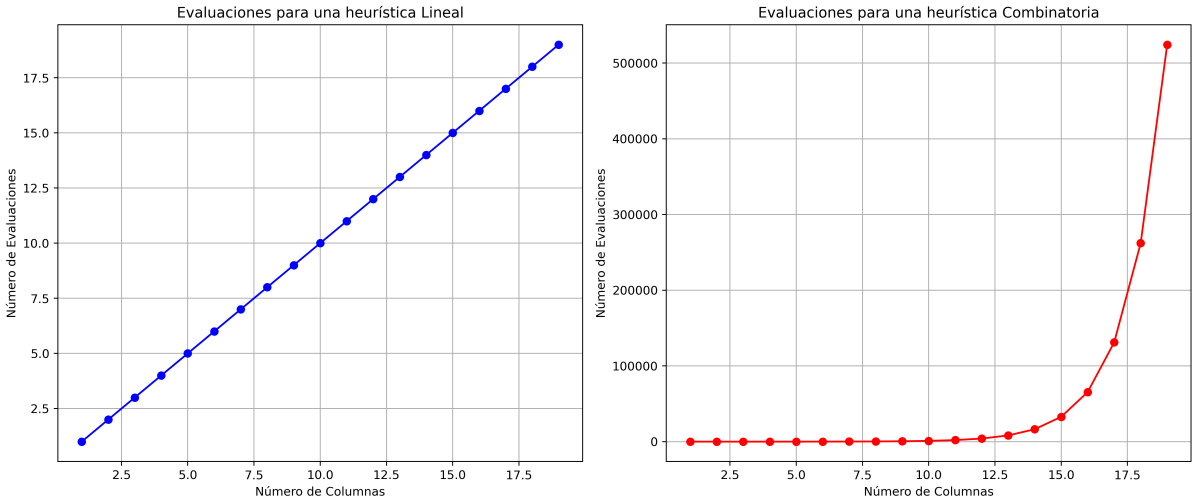

Figura 7: Comparación de la cantidad de evaluaciones a realizar en función de la cantidad de columnas dependiendo si la heurística evalúa la columna como un todo, o si evalúa todas las combinaciones posibles

Mejoras para reducir la cantidad de cálculos en heurísticas con combinatorias de residuos

Con el fin de optimizar la velocidad de ejecución de las heurísticas que utilizan las combinatorias, se implementa una agrupación por combinaciones repetidas. De esta forma, se ejecuta una sola vez la evaluación de una par de residuos y luego se multiplica su resultado por la cantidad de combinaciones iguales (algoritmo 9).

Algoritmo 9 Optimización al evaluar las combinaciones dentro de una columna

Require: *columna*

1: $combinaciones\_agrupadas \leftarrow Contar(combinaciones(columna, 2))$

2: $puntaje\_columna \leftarrow 0$

3: **for** cada $(par\_de\_residuos, repeticiones)$ en $combinaciones\_agrupadas$ **do**

4: $\quad puntaje\_columna \leftarrow puntaje\_columna + (evaluar[par\_de\_residuos] \times repeticiones)$

5: **end for**

6: **return** $puntaje\_columna$

Donde la función **Contar** se encarga de agrupar combinaciones idénticas en una tupla del tipo [pares de residuos, repeticiones].

Mejoras para reducir la cantidad de cálculos por columna

Gracias a la naturaleza de todas las heurísticas elaboradas en este trabajo, no hay diferencia alguna para el algoritmo genérico del cálculo de la energía independientemente del orden de sus elementos. El resultado de cualquier función objetivo será para todas las combinaciones de orden de sus elementos (figura 8).

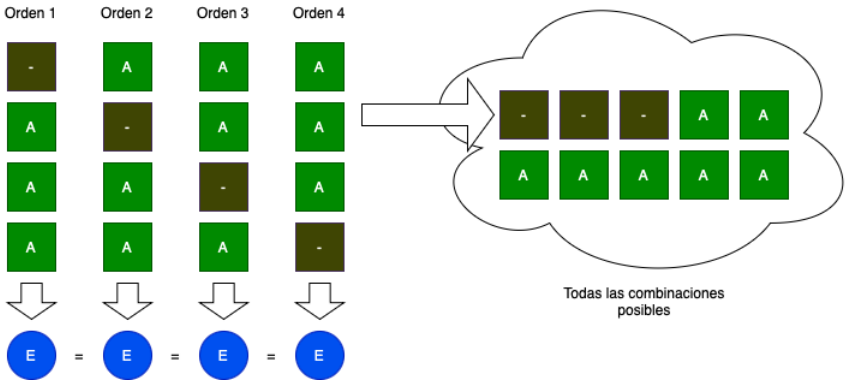

Figura 8: Todos los ordenamientos posibles dentro de una columna conducen al mismo puntaje de energía. Incluso cuando se hacen las combinaciones dentro de una columna, el orden no altera las combinaciones halladas.

De esta forma, se puede aplicar la técnica de *"caching"* en Python para guardar el resultado de la función de energía de la cada columna siempre que se pase como parámetro la columna ordenada.

3.4. Modificaciones a SA: MSASA

Con el objetivo de mejorar los alineamientos obtenidos por SA se altera el algoritmo original agregando dos escenarios totalmente configurables desde los parámetros del *software*.

- **Cantidad de cambios por iteración**: al contrario de la definición SA original, se permite que el estado vecino tenga más de un cambio. Por defecto, la cantidad es 1, manteniendo la forma original del algoritmo SA.
- **Obtener el mejor estados vecino**: se define una cantidad de estados vecinos a evaluar por cada iteración del ciclo principal. En caso de SA original, esta cantidad es 1.
- **Modo estricto y modo libre de comparación de energía**: Mediante esta opción se altera la forma en que se selecciona a un nuevo estado como el estado actual del sistema. El modo libre es el modo original de SA.

Todas estas modificaciones pueden habilitarse o deshabilitarse para permitir que la ejecución de SA sea como fue definido por sus autores originalmente.

3.4.1. Cantidad de cambios por iteración

El programa cuenta con un parámetro para definir hasta cuantos cambios por iteración se hacen sobre un estado. Este valor determina cuantas adiciones y deleciones tendrá un nuevo estado a evaluar (algoritmo 10). Dado que el parámetro tiene define una cantidad máxima, al momento de ejecutar cada iteración de SA, se obtiene al azar un número de cambios entre 1 y el parámetro elegido por el usuario.

Algoritmo 10 Alteración del estado actual

Require: secuencias

Require: cantidad_de_cambios

```
1: nuevas_secuencias ← copia(secuencias)
2: for contador_de_cambios en rango(aleatorio(1, cantidad_de_cambios)) do
3:     nuevas_secuencias ← generar_nuevo_estado(nuevas_secuencias)
4: end for
5: return nuevas_secuencias
```

3.4.2. Obtener el mejor estados vecino

Esta variante de SA genera más de un estado vecino a partir del estado actual, para luego elegir al mejor de ellos para ser evaluado a fin de saber si puede o no reemplazar al actual (algoritmo 11). Esta técnica está hibridada con la sección anterior, de forma que para cada estado vecino, se aplican una o más modificaciones.

Algoritmo 11 Determinación del mejor estado vecino

Require: secuencias
Require: cantidad_de_cambios
Require: cantidad_de_estados_vecinos

```
1: nuevas_secuencias ← copia(secuencias)
2: mejor_estado_vecino ← Nulo
3: mejor_energía_estado_vecino ← Nulo
4: for contador_de_vecinos en rango(cantidad_de_estados_vecinos) do
5:     nuevas_secuencias ← copia(secuencias)
6:     for contador_de_cambios en rango(aleatorio(1, cantidad_de_cambios)) do
7:         nuevas_secuencias ← generar_nuevo_estado(nuevas_secuencias)
8:     end for
9:     energía_estado_vecino ← función_objetivo(nuevas_secuencias)
10:    if mejor_energía_estado_vecino == Nulo or mejor_energía_estado_vecino > mejor_energía_estado_vecino then
11:        mejor_estado_vecino ← nuevas_secuencias
12:        mejor_energía_estado_vecino ← energía_estado_vecino
13:    end if
14: end for
15: return nuevas_secuencias
```

3.4.3. Modo estricto y modo libre de comparación de energía

Esta última variación al código original SA cambia la forma en que se comparan dos estados.

- **Libre**: Se compara la energía del nuevo estado contra la energía del estado actual. Es el modo original de SA.
- **Estricto**: Se compara la energía del nuevo estado contra la energía del mejor estado hasta el momento. Esta decisión implica el riesgo de encontrar soluciones dentro de un máximo

local, pero garantizan que cada cambio de estado, sea mejor en términos de puntación hasta el momento (algoritmo 12).

Algoritmo 12 Modo libre o modo estricto

Require: modo

Require: energía_actual

Require: energía_mejor

Require: nueva_energía

1: **if** modo == *libre* **then**

2: diferencia_energía ← nueva_energía - energía_actual

3: **else**

4: diferencia_energía ← nueva_energía - energía_mejor

5: **end if**

6: **return** diferencia_energía

3.4.4. SA con todas las modificaciones

Finalmente, al aplicar los cambios, el algoritmo SA modificado difiere del original pero puede mediante la parametrización del *software* comportarse como el SA original (algoritmo 13). Esta versión del algoritmo es llamada MSASA a lo largo de este projecto.

En cada iteración del proceso de enfriamiento se genera una línea de registro a fin de tener un seguimiento del proceso SA. Al finalizar la ejecución, dichas líneas, previamente guardadas en un archivo, son utilizadas para crear gráficos de diagnóstico. De esta forma se observan las evoluciones de temperatura, iteraciones, tiempo, largo de las secuencias y eventos de cambio de estado (figura 9 para el modo libre y figura 10 para el modo estricto).

Algoritmo 13 Función Simulated Annealing modificado

Require: secuencias, temperatura_actual, tasa_de_enfriamiento, temperatura_min, limite_sin_cambios, cantidad_de_cambios, cantidad_de_estados_vecinos, modo

```
iteración ← 0
sin_cambios ← 0
energía_actual ← función_objetivo(secuencias)
energía_mejor ← energía_actual
while temperatura_actual > temperatura_min do
    energía_iteración ← energía_actual

    for contador_de_vecinos en rango(cantidad_de_estados_vecinos) do

        nuevas_secuencias ← copia(secuencias)
        for contador_de_cambios en rango(aleatorio(1, cantidad_de_cambios)) do
            nuevas_secuencias ← generar_nuevo_estado(nuevas_secuencias)
        end for
        energía_estado_vecino ← función_objetivo(nuevas_secuencias)
        if mejor_energía_estado_vecino == Nulo or mejor_energía_estado_vecino > mejor_energía_estado_vecino then
            mejor_estado_vecino ← nuevas_secuencias
            mejor_energía_estado_vecino ← energía_estado_vecino
        end if
    end for

    if modo == libre then
        acepta_estado ← debe_aceptar(energía_actual, mejor_energía_estado_vecino, temperatura_actual)
    else
        acepta_estado ← debe_aceptar(energía_mejor, mejor_energía_estado_vecino, temperatura_actual)
    end if
    if acepta_estado == Verdadero then
        secuencias ← copiar(nuevo_estado)
        energía_actual ← nueva_energía
    end if
    if mejor_energía_estado_vecino > energía_mejor then
        energía_mejor ← mejor_energía_estado_vecino
    end if
    if energía_iteración == energía_actual then
        sin_cambios ← sin_cambios + 1
    else
        sin_cambios ← 0
    end if
    temperatura_actual ← temperatura_actual × tasa_de_enfriamiento
    iteración ← iteración + 1
    if sin_cambios ≥ limite_sin_cambios then
        break                                   ▷ Interrumpir el ciclo de iteraciones
    end if
end while
return secuencias
```

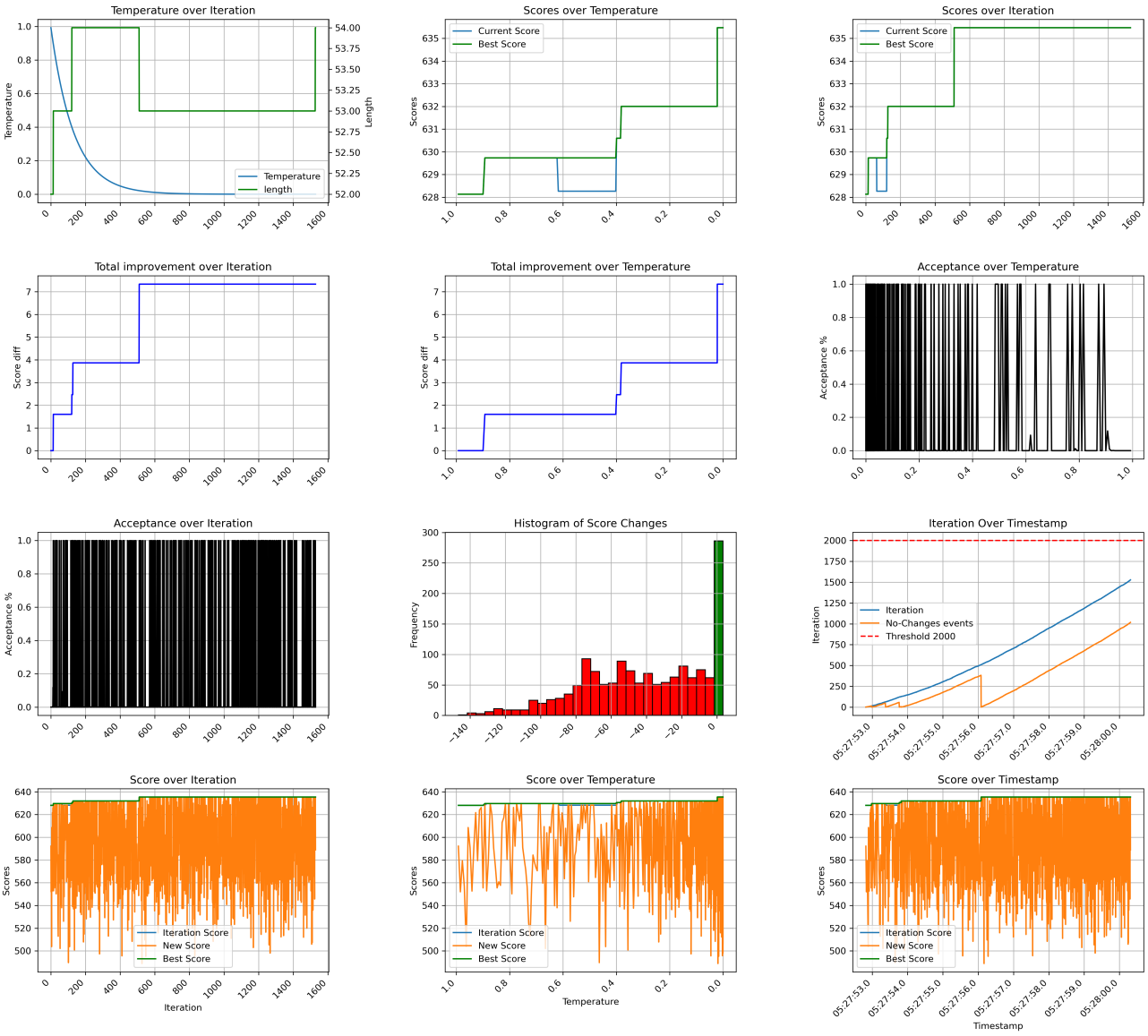

Figura 9: Gráficas diagnóstico para la confección del MSA del caso BBS12014 usando la heurística Similitud Blosum62 en modo libre.

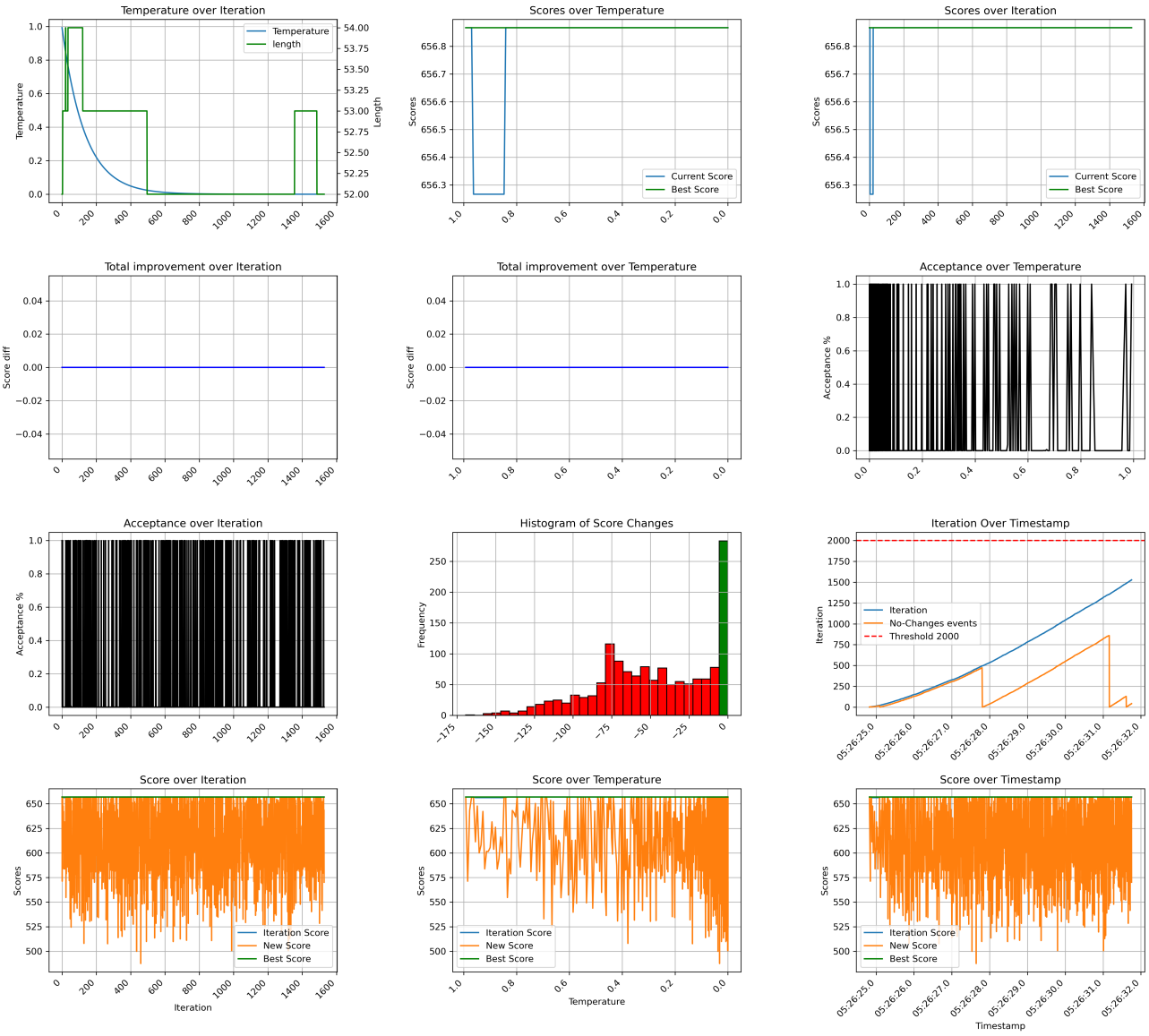

Figura 10: Gráficas diagnóstico para la confección del MSA del caso BBS12014 usando la heurística Similitud Blosum62 en modo estricto.

3.5. Funciones heurísticas aplicadas al cálculo de la energía de un estado

Como se explica en las secciones anteriores, en el contexto de la resolución del problema MSA, la energía de un estado es la medida de calidad de dicho alineamiento múltiple. En este trabajo el objetivo del algoritmo SA es maximizar la función de energía. La premisa consiste en afirmar que a mayor energía, mejor es el resultado del alineamiento final.

- **Heurística "Coincidencias"**: Cuenta grupos de caracteres por columna, aplicando penalizaciones por *gaps*.

- **Heurística "Identidad"**: Busca optimizar la proporción de columnas cuyas elementos son todos iguales, aplicando penalizaciones por *gaps*.
- **Heurística "Similitud Blosum62"**: Utiliza la matriz de sustitución Blosum62 para evaluar la divergencia entre los amino ácidos de cada columna. Para cada columna se evaluan todas las combinaciones de dos residuos. En caso de encontrar un *gap* se aplica una penalización. La matriz fue diseñada para encontrar alineamientos locales.
- **Heurística "Similitud PAM250"**: Similar a la función anterior pero utilizando la matriz PAM250. Teóricamente, esta matriz es mejor para obtener alineamientos globales.
- **Heurística "Similitud Gonnet92"**: Similar a la función anterior pero utilizando la matriz GONNET92. Teóricamente, esta matriz fue concebida para alineamientos *glocales*: tanto locales como globales.
- **Heurística "Global"**: Emula el comportamiento del algoritmo *Needleman-Wunsch* columna por columna.
- **Heurística "Local"**: Similar a la función anterior, pero busca replicar el comportamiento del algoritmo *Smith-Waterman* al ser más permisivo con los *gaps*.

En las pruebas realizadas se utilizan los parámetros como se describe a continuación (tabla 16).

Nombre	**Coin**	**No coin**	**Gap**	**T_{ini}**	**T_{min}**	**Enfr**	**Alt por it**	**Vec por it**
Identidad	8	0	1	5	0.00009	0.99	10	5
Coincidencias	10	0.75	0.10	0.90	0.00001	0.99	10	10
Blosum62	NA	-4.0	-4.0	1.0	0.00001	0.9925	10	5
PAM250	NA	-8.0	-8.0	1.0	0.00001	0.9925	10	5
Gonnet92	NA	-4.0	-6.0	1.0	0.00001	0.995	10	5
Global	10.0	-1.0	-4.0	0.80	0.00005	0.995	10	5
Local	7.0	1.0	-3.0	0.80	0.00009	0.995	10	5

Tabla 16: Comparación de los parámetros para todas las heurísticas. El encabezado de la tabla se corresponde con las columnas "Puntos por Coincidencia", "Puntos por No-Coincidencia", "Penalidad por *gap*", "Temperatura inicial", "Temperatura mínima o final", "Tasa de enfriamiento", "Cantidad de alteraciones máxima a realizar en cada estado vecino", "Cantidad de vecinos a evaluar por cada iteración para obtener el mejor vecino".

3.5.1. Cálculo de energía para la heurística "Coincidencias"

El objetivo de esta heurística es encontrar un alineamiento donde cada columna tiene la menor cantidad de sub-grupos de caracteres. Idealmente, cada columna tiene que tener un solo grupo de caracteres. Si en una columna el grupo con mayor cantidad de repeticiones es de un *gap*, entonces se usa la penalidad por *gap*. Sino, por cada grupo con mayor frecuencia (puede haber uno o más grupos) se multiplica por el puntaje por coincidencia (algoritmo 14).

Algoritmo 14 Contar Coincidencias

Require: secuencias, puntaje_por_coincidencia, penalidad_por_gap

```
coincidencias ← 0
for columna en columnas(secuencias) do
    conteos_de_caracteres ← contar_frecuencias(columna)
    mayor_frecuencia ← máximo(conteos_de_caracteres)
    if carácter_más_común(conteos_de_caracteres) == gap then
        coincidencias ← coincidencias + (penalidad_por_gap × mayor_frecuencia)
    else
        for frecuencia in conteos_de_caracteres con frecuencia == mayor_frecuencia do
            coincidencias ← coincidencias + (puntaje_por_coincidencia × mayor_frecuencia)
        end for
    end if
end for
return coincidencias
```

En este caso, dada una columna dentro C_m con n residuos que pueden ser un aminoácido o símbolos de *gap* (tabla 17):

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
$SECUENCIA_a$	M	V	Y	M	-	-	M	Y
$SECUENCIA_b$	M	I	Y	M	-	M	-	I
$SECUENCIA_c$	M	I	Y	I	-	M	-	I
$SECUENCIA_d$	M	I	Y	I	-	M	-	I
$SECUENCIA_e$	M	I	Y	T	-	M	T	I
$SECUENCIA_n$	M	V	V	S	-	M	M	I
Energía	**+6**	**+4**	**+5**	**+4**	**-6**	**+5**	**-3**	**+5**

Tabla 17: Pequeño ejemplo teórico de *6* secuencias y *8* columnas utilizando la estrategia de conservación. En cada columna se sombrean los grupos con mayor frecuencia. En C_4 hay dos grupos.

Luego que se obtiene la energía total se la divide por la cantidad de columnas. Esto se realiza para evitar que la matriz crezca sin control en número de columnas. Para el ejemplo de la tabla 17, el resultado final es:

$$\frac{\sum C_1,...,C_8}{8} = \frac{20}{8} = 2{,}5$$

3.5.2. Cálculo de energía para la heurística "Identidad"

El objetivo de esta heurística es encontrar un alineamiento donde cada columna tiene la menor cantidad de sub-grupos de caracteres. Idealmente, cada columna tiene que tener un solo grupo de caracteres. Si en una columna el grupo con mayor cantidad de repeticiones es de un *gap*, entonces se usa la penalidad por *gap*. Sino, por cada grupo con mayor frecuencia (puede haber uno o más grupos) se multiplica por el puntaje por coincidencia (algoritmo 15).

Algoritmo 15 Contar Coincidencias

Require: secuencias, puntaje_por_coincidencia, puntaje_de_no_coincidencia, penalidad_por_gap

```
columnas_idénticas ← 0
for col en columnas(secuencias) do
    if todos(columna == gap) then
        columnas_idénticas ← columnas_idénticas + penalidad_por_gap
    else
        if todos(columna == columna[0]) then
            columnas_idénticas ← columnas_idénticas + puntaje_de_coincidencia
        else
            columnas_idénticas ← columnas_idénticas + puntaje_de_no_coincidencia
        end if
    end if
end for
porcentaje_de_identidad ← (columnas_idénticas / total_columnas)
return porcentaje_de_identidad
```

En este caso, dada una columna dentro C_m con n residuos que pueden ser un aminoácido o símbolos de *gap* (tabla 18).

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
$SECUENCIA_a$	M	V	Y	A	-	-	M	Y
$SECUENCIA_b$	M	I	Y	A	-	M	-	I
$SECUENCIA_c$	M	I	Y	A	-	M	-	I
$SECUENCIA_d$	M	I	Y	A	-	M	-	I
Energía	+2	+1	+2	+2	-2	+1	+1	+1

Tabla 18: Pequeño ejemplo teórico de *4* secuencias y *8* columnas utilizando la estrategia de Identidad. El puntaje por columnas idénticas es 2.0, mientras que en caso de no-coincidencias es 1.0 y la penalidad por *gap* es -2.0

Luego que se obtiene la energía total se la divide por la cantidad de columnas. Esto se realiza para evitar que la matriz crezca sin control en número de columnas. Para el ejemplo de la tabla 18, el resultado final es:

$$\frac{\sum C_1, ..., C_8}{8} = \frac{8}{8} = 1$$

3.5.3. Cálculo de energía para la heurística "Similitud Blosum62"

El objetivo de esta heurística es encontrar un alineamiento donde cada columna tiene la menor cantidad de sub-grupos de caracteres. Idealmente, cada columna tiene que tener un solo grupo de caracteres. Si en una columna el grupo con mayor cantidad de repeticiones es de un *gap*, entonces se usa la penalidad por *gap*. Sino, por cada grupo con mayor frecuencia (puede haber uno o más grupos) se multiplica por el puntaje por coincidencia (algoritmo 16).

Algoritmo 16 Similitud usando Blosum62

Require: secuencias, matriz_blosum62, penalidad_por_gap

```
1:  puntaje_de_similitud ← 0
2:  for columna en columnas(secuencias) do
3:      for cada (res1, res2) en combinaciones(columna, 2) do
4:          if res1 == gap or res2 == gap then
5:              puntaje_de_similitud ← puntaje_de_similitud + penalidad_por_gap
6:          else
7:              puntaje_de_similitud ← puntaje_de_similitud + matriz_blosum62[(res1, res2)]
8:          end if
9:      end for
10: end for
11: return puntaje_de_similitud
```

En este caso, dada una columna dentro C_m con n residuos que pueden ser un aminoácido o símbolos de *gap* (tabla 19):

	C_1
$SECUENCIA_a$	M
$SECUENCIA_a$	V
$SECUENCIA_a$	-
$SECUENCIA_a$	L

Tabla 19: Pequeño ejemplo teórico de *4* secuencias y *1* columna utilizando la estrategia de Similitud.

Las combinaciones a buscar dentro de la matriz de substituciones son las siguientes 6:

1. `[M, V]`: 1.0
2. `[M, -]`: -8.0 (penalidad por *gap*)
3. `[M, L]`: 2.0
4. `[V, -]`: -8.0 (penalidad por *gap*)
5. `[V, L]`: 1.0
6. `[L, -]`: -8.0 (penalidad por *gap*)

Obteniendo finalmente -20.0 puntos antes de normalizar. Todas las heurísticas son normalizadas entre 0 y 1 teniedno en cuenta los puntos de no-coincidencia y la penalidad por *gap*.

3.5.4. Cálculo de energía para la heurística "Similitud PAM250"

Similar al método anterior utilizando la matriz PAM250:

1. `[M, V]`: 1.0
2. `[M, -]`: -8.0 (penalidad por *gap*)
3. `[M, L]`: 4.0
4. `[V, -]`: -8.0 (penalidad por *gap*)
5. `[V, L]`: 2.0
6. `[L, -]`: -8.0 (penalidad por *gap*)

Obteniendo finalmente -17.0 puntos antes de normalizar.

3.5.5. Cálculo de energía para la heurística “Similitud Gonnet92”

Similar al método anterior utilizando la matriz Gonnet92:

1. `[M, V]`: 1.6
2. `[M, -]`: -8.0 (penalidad por *gap*)
3. `[M, L]`: 2.8
4. `[V, -]`: -8.0 (penalidad por *gap*)
5. `[V, L]`: 1.8
6. `[L, -]`: -8.0 (penalidad por *gap*)

Obteniendo finalmente -17.8 puntos antes de normalizar.

3.5.6. Cálculo de energía para la heurística “Global”

Este algortimo es muy similar a los anteriores al utilizar también una matriz de substitución creada en base los puntajes de coincidencia, no coincidencia y la penalidad de *gap* (algoritmo 17). Por cada columna se hacen combinaciones de dos elementos y cada una de ellas es evaluada de acuerdo a sus residuos. Si alguno de ellos es un *gap* se suma la penalidad, sino, si son iguales, se suma el puntaje de coincidencia. En otro caso, se suma el puntaje de no-coincidencia. El resultado final de todas las columnas es divido por la cantidad de columnas de la matriz alineada para intentar que no crezca horizontalmente sin control.

Algoritmo 17 Maximizar global

Require: secuencias

Require: puntaje_por_coincidencia

Require: puntaje_por_no_coincidencia

Require: penalidad_por_gap

```
puntaje_de_calidad ← 0
for columna en columnas(secuencias) do
    puntajes_de_columna ← 0
    for cada (res1, res2) en combinaciones(columna, 2) do
        if res1 == gap or res2 == gap then
            puntaje_combinación ← penalidad_por_gap
        else
            if columna[i] == columna[j] then
                puntaje_combinación ← puntaje_por_coincidencia
            else
                puntaje_combinación ← puntaje_por_no_coincidencia
            end if
        end if
        puntajes_de_columna ← puntajes_de_columna + puntaje_combinación
    end for
    puntaje_de_calidad ← puntaje_de_calidad + puntajes_de_columna
end for
return puntaje_de_calidad
```

En este caso, dada una columna dentro C_m con n residuos que pueden ser un aminoácido o símbolos de *gap* (tabla 20):

	C_1
$SECUENCIA_a$	M
$SECUENCIA_a$	V
$SECUENCIA_a$	-
$SECUENCIA_a$	M

Tabla 20: Pequeño ejemplo teórico de *4* secuencias y *1* columna utilizando la estrategia de Similitud.

Las combinaciones a buscar dentro de la matriz de substituciones son las siguientes 6:

1. `[M, V]`: puntos de no-coincidencia
2. `[M, -]`: penalidad por *gap*
3. `[M, M]`: puntos de coincidencia
4. `[V, -]`: penalidad por *gap*
5. `[V, M]`: puntos de no-coincidencia
6. `[-, M]`: penalidad por *gap*

3.5.7. Cálculo de energía para la heurística "Local"

Idéntico al algoritmo anterior pero intenta ser más benevolente con los gaps al otorgar menos puntos de penalidad y además sumar un poco más de puntos por no-coincidencia que su homólogo "Global".

4. Diseño experimental

Con el fin de corroborar las hipótesis planteadas al inicio de este trabajo, se procede a detallar lo relacionado a la selección de las secuencias de prueba, la determinación de los escenarios de prueba, la elección de las herramientas a comprar, la configuración del algoritmo propuesto en este trabajo para resolver MSA, denominado “MSASA”.

Dentro de este capítulo, también, se explica la elección de diferentes herramientas para obtener métricas de calidad de un MSA. En general, este tipo de herramientas necesitan conocer de antemano el MSA de referencia para compararlo con los resultados obtenidos. Al momento de validar la tres hipótesis planteadas, se utilizan grupo de pruebas a partir de un sub-conjunto de una base de datos de secuencias curadas a mano por un grupo de investigación. Además de las secuencias a alinear, la base de datos provee un *software* de comparación que permite determinar que tan bien fue el rendimiento de una aplicación MSA.

En vista de la naturaleza estocástica del algoritmo SA, todas las pruebas se ejecutan 30 veces con el fin de obtener resultados independientes de una ejecución única. Se utiliza una máquina virtual que garantiza en todo momento que el hardware es el mismo y que no se degrada con el tiempo por factores externos ni debido al uso del mismo al ejecutar las pruebas:

- vCPU: 4 núcleos Intel de arquitectura x86_64
- RAM: 16 GB
- Espacio de almacenamiento: 80 GB
- Sistema operativo: Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-102-generic x86_64)

Dicha máquina virtual es provista por la empresa Hetzner Cloud y se encuentra físicamente dentro del *datacenter* “hel1-dc2” en Finlandia.

4.1. Selección de las secuencias de prueba

En este trabajo se utiliza la base de datos de secuencias llamada *BAli BASE* (*Benchmark Alignment dataBASE*) [61]. Esta base de datos es utilizada ampliamente por la comunidad de Bioinformática para evaluar y comparar diferentes MSA. Los alineamientos de referencias incluidos en dicha base da datos se basan en estructuras tridimensionales similares, además de incluir motivos lineales (*LM* por *linear motifs* en inglés).

4.1.1. Secuencias de prueba provenientes de *BAli BASE*

Cada conjunto de secuencias a alinear provisto por *BAli BASE* es acompañado por un alineamiento de referencia de alta calidad curada por personas expertas en alineación de secuencias. Luego, los alineamientos que se generan utilizando diferentes piezas de *software* MSA son comparados y evaluados mediantes métricas de calidad . Algunas de estas métricas utilizan el alineamiento de referencia, mientras que otras no lo requieren.

Sobre regiones altamente conservadas (*core blocks*)

Los *core blocks* son regiones alineadas en todas las secuencias del conjunto de datos de prueba, es decir, son fragmentos altamente conservados dentro de un MSA y se utilizan para evaluar la calidad del alineamiento en el contexto de *BAli BASE*. Estos bloques son regiones que se consideran esenciales para la función biológica de la proteína a alinear. Al evaluar solamente *core blocks*, el *software* evaluador que acompaña la base de datos de *BAli BASE* (*BAli Score*) puede hacer una evaluación más precisa del rendimiento de un alineamiento efectuado por un *software* de MSA. Al contrario de los bloques altamente conservados, las regiones que no están alineadas en todas las secuencias son conocidas como *gaps*; en tanto que, las regiones que están alineadas en algunas pero no en todas las secuencias se denominan *regiones de divergencia*.

Sobre los motivos lineales

Los motivos son secuencias de residuos cortas y altamente conservadas que se repite en una o varias secuencias. Los motivos se caracterizan por desempeñar una función biológica específica, como la regulación de la expresión génica o la interacción proteína-proteína. En general estos motivos se encuentra en regiones desordenadas de las proteínas, haciendo que alinearlas sea más difícil para los métodos clásicos. La mayoría de los motivos lineales tienen una longitud de tres a diez aminoácidos con al menos un residuo variable o ambiguo. Debido a su naturaleza, estos motivos son complicados de distinguir entre ellos y regiones aleatorias.

4.1.2. Grupos de referencia

Esta base de datos se organiza en diez grupos de referencia de secuencias con diferentes grados de similitud y complejidad:

- **Grupo de referencia número 1**: Alineamientos equidistantes de secuencias de similar longitud sin inserciones o extensiones. Este grupo contiene regiones altamente conser-

vadas que solo incluyan regiones de las secuencias que pueden ser alineadas con alta confianza.

- **Grupo de referencia número 2**: Alineamientos de hasta 3 secuencias con menos de 25 por ciento de identidad provenientes del grupo de referencia anterior, con al menos una familia de hasta 15 secuencias próximas. Este grupo contiene regiones altamente conservadas que solo incluyen regiones de las secuencias que pueden ser alineadas con alta confianza.

- **Grupo de referencia número 3**: Alineamientos de hasta 4 sub-grupos con hasta 25 por ciento de identidad entre secuencias de diferentes sub-grupos. Estos alineamientos son construidos mediante la incorporación de secuencias homólogas a secuencias distantes del primer grupo de referencia. Este grupo contiene regiones altamente conservadas que sólo incluyan regiones de las secuencias que pueden ser alineadas con alta confianza.

- **Grupo de referencia número 4**: Agrupa 2 categorías de alineamientos de hasta 20 secuencias que incluyen las regiones terminal *N/C* de hasta 400 residuos e inserciones de hasta 100 residuos. Este grupo contiene regiones altamente conservadas que solo incluyan regiones de las secuencias que pueden ser alineadas con alta confianza.

- **Grupo de referencia número 5**: Alineamientos con grandes inserciones internas. Este grupo contiene regiones altamente conservadas que solo incluyan regiones de las secuencias que pueden ser alineadas con alta confianza.

- **Grupos de referencia número 6, 7 y 8**: La segunda versión de *BAli BASE* incluye estos 3 grupos de referencia con proteínas transmembrana, otras con repeticiones y finalmente secuencias con permutaciones circulares. Todas las secuencias fueron escogidas o a partir de familias de proteínas de la base de datos *PFam*, o de la literatura, como así también a partir de secuencias encontradas luego de ejecutar Blast para encontrar secuencias similares.

- **Grupo de referencia número 9**: Este grupo incluye alineamientos de referencia de familias de proteínas con motivos lineales.

- **Grupo de referencia número 10**: Familias complejas de largas proteínas escogidas que buscan poner a prueba los siguientes desafíos:

(i) El foco principal siempre estuvo puesto en hallar patrones conservados en la gran mayoría de las secuencias, dejando de lado patrones menos frecuentes que pueden indicar contextos específicos de sub-familias.

(ii) Las regiones naturalmente desordenadas o largamente no estructuradas, especialmente en eucariotas, contienen motivos de vital importancia para procesos biológicos como señalización, sitios de modificaciones postraduccionales.

(iii) Si bien las técnicas de secuenciación masiva permiten incorporar gigantescos volúmenes de datos, pueden incluir errores propios del proceso o proveer información fragmentada que afectan a la calidad del alineamiento final.

4.2. Determinación de los escenarios de prueba

Se han elegido tres dimensiones de estudio al momento de crear los escenarios de prueba: la longitud de las secuencias, la cantidad de secuencias a alinear y el porcentaje de identidad que posee el alineamiento de referencia. En todos los casos se han elegido secuencias de proteínas, es decir, que las secuencias contienen aminoácidos, en consecuencia, son cadenas de texto con variaciones de veinte posibles valores más el símbolo de espacio (*gap*) que es representado por el carácter guión - [62].

4.2.1. Longitud de las secuencias

Respecto a la longitud de las cadenas, las mismas se clasifican en cortas, medianas y largas teniendo en cuenta la cantidad de residuos:

- **Secuencias de corta longitud (CL)**: Hasta de 100 residuos.
- **Secuencias de mediana longitud (ML)**: Entre 100 y 400 residuos.
- **Secuencias de larga longitud (LL)**: Más de 400 residuos.

4.2.2. Cantidad de secuencias

En este trabajo se considera que un alineamiento a realizar es pequeño si tiene menos de 100 secuencias a alinear, mientras que si tiene más se considera que posee una alta cantidad.

- **Baja cantidad de secuencias a alinear (BC)**: Menos de 100 secuencias en el archivo de entrada.

- **Alta cantidad de secuencias a alinear (AC)**: Más de 100 secuencias en el archivo de entrada.

4.2.3. Porcentaje de identidad

El porcentaje de identidad es una dimensión de estudio muy importante, ya que se utiliza para evaluar la calidad de un MSA y se refiere a la identidad o similitud entre las secuencias alineadas. A mayor valor de identidad, mayor es la probabilidad de que las secuencias tengan un origen común y sean funcionalmente similares. Gracias a esto, se determinan familias de proteínas o secuencias homologas que fueron divergiendo gracias a los procesos evolutivos. En las bases de datos de pruebas utilizadas para comparar los distintos *software* de MSA, la identidad es una medida conocida de antemano.

Por regla general, se dice que más de treinta por ciento de identidad en más de cien residuos es suficiente para que dos secuencias sean homólogas a lo largo de sus longitudes completas. Pero a veces este porcentaje puede perder algunos homólogos, dado que existen secuencias homologas con menos del treinta por ciento.

4.2.4. Casos de pruebas elegidos

A partir de las categorías de las dos primeras dimensiones, se obtienen seis grupos de pruebas que son utilizados al momento de definir los escenarios:

- **Grupo 1: CL - BC**:

  Hasta 100 residuos por secuencia.

  Hasta 100 secuencias por archivo FASTA.

- **Grupo 2: CL - AC**:

  Hasta 100 residuos por secuencia.

  Más de 100 secuencias por archivo FASTA.

- **Grupo 3: ML - BC**:

  Entre 100 y 400 residuos por secuencia.

  Hasta 100 secuencias por archivo FASTA.

- **Grupo 4: ML - AC**:

Entre 100 y 400 residuos por secuencia.

Más de 100 secuencias por archivo FASTA.

- **Grupo 5: LL - BC**:

  Más de 400 residuos por secuencia.

  Hasta 100 secuencias por archivo FASTA.

- **Grupo 6: LL - AC**:

  Más de 400 residuos por secuencia.

  Más de 100 secuencias por archivo FASTA.

Al clasificar los grupos de referencia de la base de datos *BAli BASE* de acuerdo a los grupos de pruebas, se observa la distribución de alineamientos por grupo de prueba (tabla 21). Esta tabla muestra por cada intersección de las dimensiones (longitud de las secuencias y cantidad de secuencias) cuantas secuencias se encuentran por cada uno de los grupos de referencia evaluados.

	CL		ML		LL	
Grupo de ref. *BAli BASE*	**BC**	**AC**	**BC**	**AC**	**BC**	**AC**
Ref. 1: Alineamientos con bajo porcentaje de identidad	18	0	81	0	65	0
Ref. 2: Alineamientos de familias de proteínas con alta divergencia de secuencias	7	0	39	0	36	0
Ref. 3: Familias de proteínas con alta divergencia de secuencias	2	0	24	4	26	4
Ref. 4: Alineamientos que incluyen regiones terminales *N/C*:	0	0	1	0	48	0
Ref. 5: Alineamientos con largas inserciones internas	0	0	8	0	23	0
Ref. 9: Alineamientos que incluyen motivos lineales	0	0	14	0	159	10
Ref. 10: Alineamientos de grandes familias complejas	1	1	39	14	123	40

Tabla 21: Clasificación de los MSA de prueba de la base de datos *BAli BASE* según los grupos de referencia.

Notese que pueden haber grupos de referencia sin secuencias para las dimensiones mencionadas. Por lo tanto, al tomar una secuencia de cada grupo de referencia, se definen los veintitrés casos de pruebas que se muestran en la tabla 22.

	CL		ML		LL	
Grupo de ref. *BAli BASE*	**BC**	**AC**	**BC**	**AC**	**BC**	**AC**
Ref. 1: Alineamientos con bajo porcentaje de identidad	C1		C2		C3	
Ref. 2: Alineamientos de familias de proteínas con alta divergencia de secuencias	C4		C5		C6	
Ref. 3: Familias de proteínas con alta divergencia de secuencias	C7		C8	C9	C10	C11
Ref. 4: Alineamientos que incluyen regiones terminales *N/C*:			C12		C13	
Ref. 5: Alineamientos con largas inserciones internas					C14	
Ref. 9: Alineamientos que incluyen motivos lineales			C15		C16	C17
Ref. 10: Alineamientos de grandes familias complejas	C18	C19	C20	C21	C22	C23

Tabla 22: Selección de los MSA de prueba extraídos de la base de datos *BAli BASE* por cada uno de los grupos de referencia.

En resumen, los 23 casos de prueba extraídos de la base de datos *BAli BASE* para utilizar a lo largo del proceso de ejecución y evaluación son los detallados en la tabla 23.

Caso de prueba	**Nombre**	**Grupo de ref.** ***BAli BASE***	**Long. máx AA**	**Cantidad**	**Dim. long.**	**Dim. cant.**
C1	BBS12014	Ref. 1	52	9	CL	BC
C2	BB11013	Ref. 1	101	5	ML	BC
C3	BB11004	Ref. 1	410	4	LL	BC
C4	BBS20002	Ref. 2	49	20	CL	BC
C5	BBS20009	Ref. 2	219	29	ML	BC
C6	BBS20007	Ref. 2	433	23	LL	BC
C7	BBS30016	Ref. 3	66	37	CL	BC
C8	BBS30017	Ref. 3	287	15	ML	BC
C9	BBS30001	Ref. 3	190	116	ML	AC
C10	BBS30015	Ref. 3	650	21	LL	BC
C11	BB30003	Ref. 3	514	142	LL	AC
C12	BB40010	Ref. 4	214	9	ML	BC
C13	BB40014	Ref. 4	609	9	LL	BC
C14	BBS50005	Ref. 5	806	11	LL	BC
C15	BOX132	Ref. 9	159	13	ML	BC
C16	BOX212	Ref. 9	625	8	LL	BC
C17	BOX122	Ref. 9	1041	174	LL	AC
C18	BBA0142	Ref. 10	78	19	CL	BC
C19	BBA0117	Ref. 10	77	460	CL	AC
C20	BBA0030	Ref. 10	165	19	ML	BC
C21	BBA0011	Ref. 10	251	102	ML	AC
C22	BBA0192	Ref. 10	799	4	LL	BC
C23	BBA0065	Ref. 10	786	100	LL	AC

Tabla 23: Listado de los casos de prueba MSA extraídos de la base de datos *BAli BASE* por cada uno de los grupos de referencia.

A continuación se visualiza como se distribuyen los casos de prueba entre las dos dimensiones establecidas previamente: la cantidad de secuencias y las longitudes de las mismas (figura 11). También se presentan las distribuciones de ambas dimensiones de grupos de acuerdo a la

longitud de las secuencias y cantidad de las mismas (figura 12).

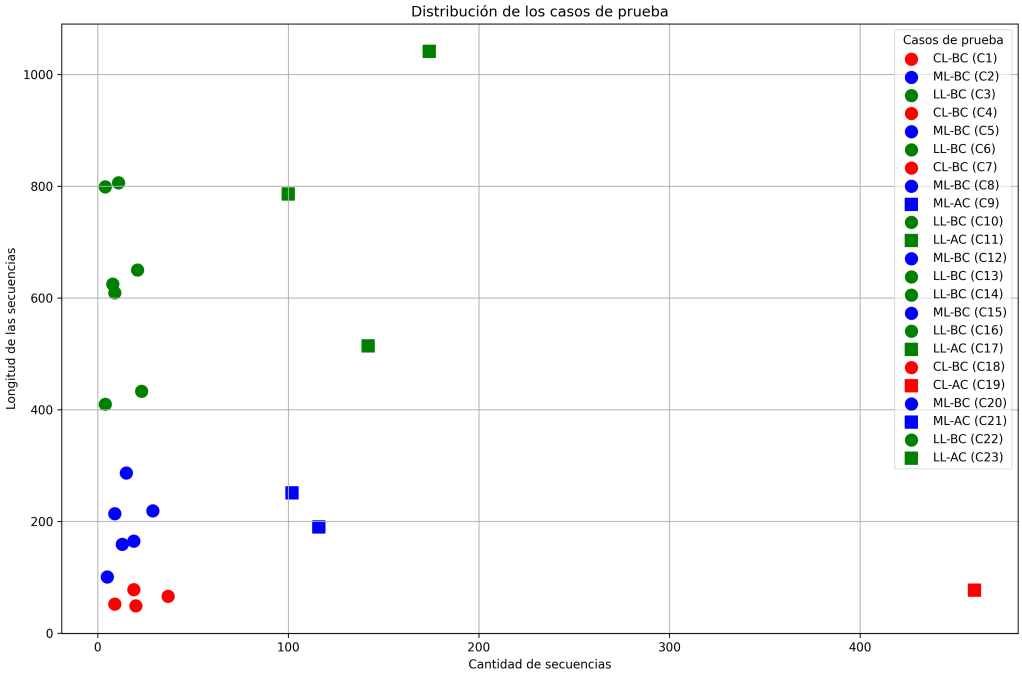

Figura 11: En rojo se marcan el grupo “CL”, en azul “ML” y en verde “LL”. Mientras que los círculos son casos de estudio del grupo “BC” y los cuadrados son “AC”.

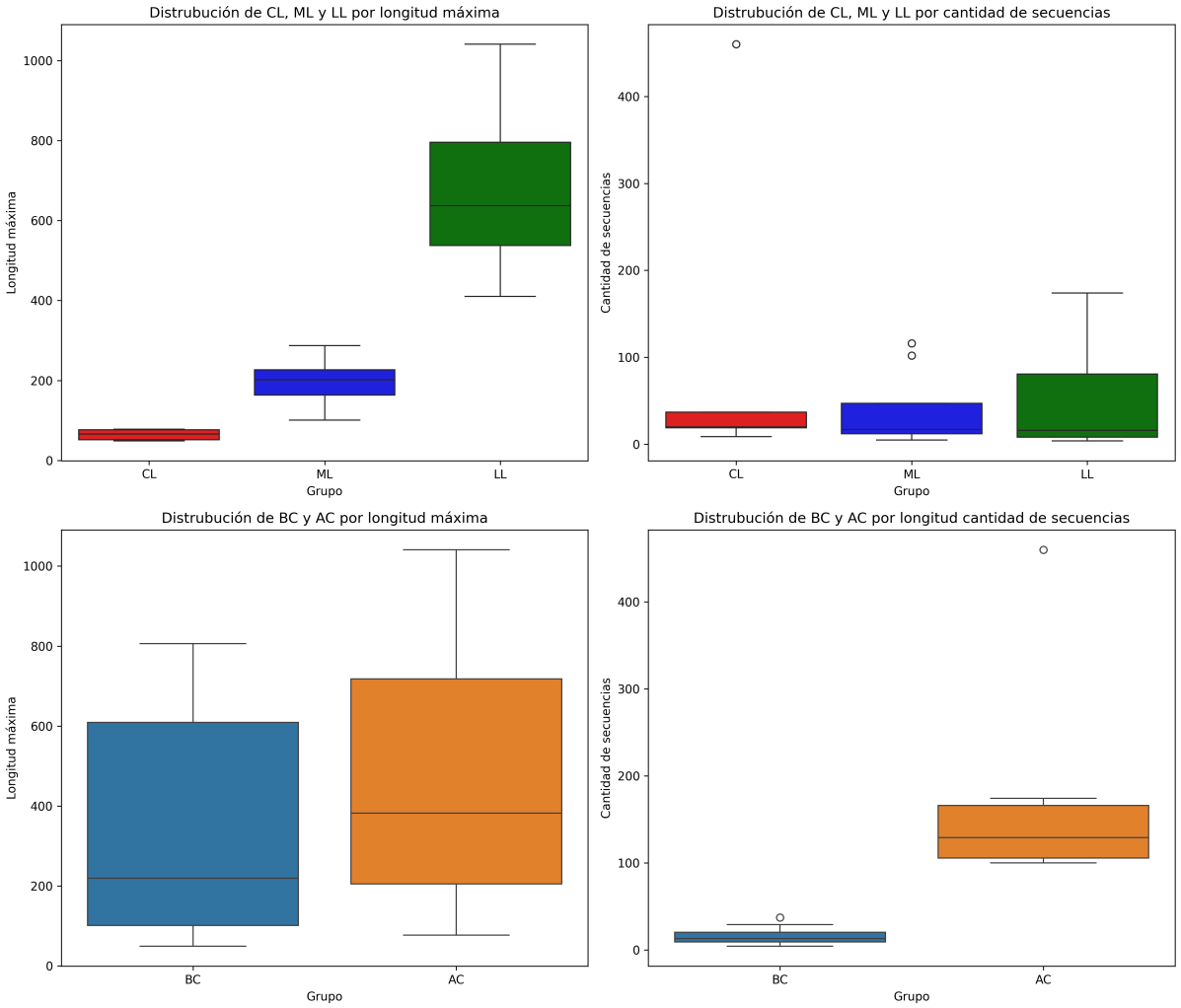

Figura 12: En rojo se marcan el grupo “CL”, en azul “ML” y en verde “LL”. Mientras que el grupo “BC” es representado en azul claro y “AC” en naranja.

4.3. Selección de las herramientas de *software* a comparar

4.3.1. Configuración de MSASA

Además de las piezas de *software* seleccionadas del estado del arte, se generan los MSA utilizando SA con modificaciones hibridado con las funciones heurísticas (funciones objetivo) mencionadas en el capítulo anterior.

Los parámetros de iteraciones y temperatura inicial se definen en base a pruebas con secuencias de BAli BASE a fin de determinar una configuración de valores que permita obtener resultados en un tiempo aceptable (menos de 20 horas por heurística).

Identidad

La idea de esta heurística es premiar a las columnas que contengan elementos idénticos. Mientras más columnas tengan solo un carácter, mejor es el puntaje final del alineamiento. Sin embargo, sin controles, el puntaje sería cada vez mejor si se incorporan columnas con *gaps*. La penalidad por *gap* busca reducir esta problemática. Incluso, se agrega un puntaje por no-coincidencias para permitir que los alineamientos puedan tener variabilidad.

En esta función objetivo, al resultado obtenido al sumar los puntajes de cada columna se lo divide por la cantidad de columnas de la matriz, es decir, la longitud máxima de residuos. Esta división intenta limitar el crecimiento de las secuencias con *gaps* inútiles.

- **Heurística "Identidad Libre"**
    - Temperatura inicial: 5.0 °C
    - Temperatura final: 0.00009 °C
    - Tasa de enfriamiento: 0.99
    - Puntos por coincidencia: 8.0
    - Puntos por no-coincidencia: 1.0
    - Penalidad por *gap*: 2.0
    - Cantidad máxima sin cambios de energía: 2000
    - Cantidad máxima de cambios en el estado vecino por iteración: 10
    - Estados vecinos a evaluar por iteración: 5
    - Modo: "Libre"
- **Heurística "Identidad Estricta"**
    - Temperatura inicial: 5.0 °C
    - Temperatura final: 0.00009 °C
    - Tasa de enfriamiento: 0.99
    - Puntos por coincidencia: 8.0
    - Puntos por no-coincidencia: 1.0
    - Penalidad por *gap*: 2.0
    - Cantidad máxima sin cambios de energía: 2000
    - Cantidad máxima de cambios en el estado vecino por iteración: 10

- Estados vecinos a evaluar por iteración: 5
- Modo: “Estricto”

Coincidencias

Con esta función se busca minimizar la cantidad de grupos de caracteres iguales dentro de una columna. Lo idea es tener un solo grupo por columna. Nuevamente, si la columna solo tiene *gaps* se suma una penalidad para intentar reducir las columnas de *gaps* en el medio de la matriz. En caso que no hay grupos porque todos los caracteres son diferentes, se suma un puntaje por no-coincidencias.

Para evitar que la matriz crezca indefinidamente, se divide el total de puntos de todas las columnas por la cantidad de columnas.

- **Heurística “Coincidencias Libre”**
  - Temperatura inicial: 0.90 °C
  - Temperatura final: 0.00001 °C
  - Tasa de enfriamiento: 0.99
  - Puntos por coincidencia: 10.0
  - Puntos por no-coincidencia: 0.75
  - Penalidad por *gap*: 0.10
  - Cantidad máxima sin cambios de energía: 2000
  - Cantidad máxima de cambios en el estado vecino por iteración: 10
  - Estados vecinos a evaluar por iteración: 10
  - Modo: “Libre”
- **Heurística “Coincidencias Estricta”**
  - Temperatura inicial: 0.90 °C
  - Temperatura final: 0.00001 °C
  - Tasa de enfriamiento: 0.99
  - Puntos por coincidencia: 10.0

- Puntos por no-coincidencia: 0.75
- Penalidad por *gap*: 0.10
- Cantidad máxima sin cambios de energía: 2000
- Cantidad máxima de cambios en el estado vecino por iteración: 10
- Estados vecinos a evaluar por iteración: 10
- Modo: “Estricto”

Similitud Blosum62

La matriz de substitución fue normalizada entre 0 y 1 teniendo en cuenta los puntajes por no-coincidencia y la penalidad por *gap*. El valor mínimo de Blosum62 es -4.0 y el máximo 11.0. Los puntos de no-coincidencia y la penalidad por *gap* son idénticas al valor mínimo de Blosum62.

En esta heurística siempre se devuelve el puntaje total por 1 para evitar interferir en el crecimiento de las columnas de la matriz de secuencias alineadas.

- **Heurística “Similitud Blosum62 Libre”**:
  - Temperatura inicial: 1.0 °C
  - Temperatura final: 0.00001 °C
  - Tasa de enfriamiento: 0.9925
  - Puntos por no-coincidencia: -4.0
  - Penalidad por *gap*: -4.0
  - Cantidad máxima sin cambios de energía: 2000
  - Cantidad máxima de cambios en el estado vecino por iteración: 10
  - Estados vecinos a evaluar por iteración: 5
  - Modo: “Libre”
- **Heurística “Similitud Blosum62 Estricta”**:
  - Temperatura inicial: 1.0 °C
  - Temperatura final: 0.00001 °C

- Tasa de enfriamiento: 0.9925
- Puntos por no-coincidencia: -4.0
- Penalidad por *gap*: -4.0
- Cantidad máxima sin cambios de energía: 2000
- Cantidad máxima de cambios en el estado vecino por iteración: 10
- Estados vecinos a evaluar por iteración: 5
- Modo: “Estricto”

Similitud PAM250

La matriz de substitución fue normalizada entre 0 y 1 teniendo en cuenta los puntajes por no-coincidencia y la penalidad por *gap*. El valor mínimo de PAM250 es -8.0 y el máximo 17.0. Los puntos de no-coincidencia y la penalidad por *gap* son idénticas al valor mínimo de PAM250.

En esta heurística siempre se devuelve el puntaje total por 1 para evitar interferir en el crecimiento de las columnas de la matriz de secuencias alineadas.

- **Heurística “Similitud PAM250 Libre”**:
  - Temperatura inicial: 1.0 °C
  - Temperatura final: 0.00001 °C
  - Tasa de enfriamiento: 0.9925
  - Puntos por no-coincidencia: -8.0
  - Penalidad por *gap*: -8.0
  - Cantidad máxima sin cambios de energía: 2000
  - Cantidad máxima de cambios en el estado vecino por iteración: 10
  - Estados vecinos a evaluar por iteración: 5
  - Modo: “Libre”
- **Heurística “Similitud PAM250 Estricta”**:
  - Temperatura inicial: 1.0 °C
  - Temperatura final: 0.00001 °C

- Tasa de enfriamiento: 0.9925
- Puntos por no-coincidencia: -8.0
- Penalidad por *gap*: -8.0
- Cantidad máxima sin cambios de energía: 2000
- Cantidad máxima de cambios en el estado vecino por iteración: 10
- Estados vecinos a evaluar por iteración: 5
- Modo: “Estricto”

Similitud Gonnet92

La matriz de substitución fue normalizada entre 0 y 1 teniendo en cuenta los puntajes por no-coincidencia y la penalidad por *gap*. El valor mínimo de Gonnet92 es -5.2 y el máximo 14.2. Los puntos de no-coincidencia y la penalidad por *gap* son diferentes al mínimo de Gonnet92, por lo que la normalización de 0 a 1 se construye tomando -6.0 y 14.2 como el rango de valores.

En esta heurística siempre se devuelve el puntaje total por 1 para evitar interferir en el crecimiento de las columnas de la matriz de secuencias alineadas.

- **Heurística “Similitud Gonnet92 Libre”**:
  - Temperatura inicial: 1.0 °C
  - Temperatura final: 0.00001 °C
  - Tasa de enfriamiento: 0.995
  - Puntos por no-coincidencia: -8.0
  - Penalidad por *gap*: -8.0
  - Cantidad máxima sin cambios de energía: 2000
  - Cantidad máxima de cambios en el estado vecino por iteración: 10
  - Estados vecinos a evaluar por iteración: 5
  - Modo: “Libre”
- **Heurística “Similitud Gonnet92 Estricta”**:
  - Temperatura inicial: 1.0 °C

- Temperatura final: 0.00001 °C
- Tasa de enfriamiento: 0.995
- Puntos por no-coincidencia: -4.0
- Penalidad por *gap*: -6.0
- Cantidad máxima sin cambios de energía: 2000
- Cantidad máxima de cambios en el estado vecino por iteración: 10
- Estados vecinos a evaluar por iteración: 5
- Modo: “Libre”

Global

La heurística intenta ser más estricta con la inclusión de *gaps* por lo que tiene una penalidad mayor ante la aparición de estos símbolos y se premian las coincidencias con un valor muy alto. De esta forma, busca simular los MSA globales.

Para evitar que la matriz de secuencias alineadas crezca indefinidamente, se divide el puntaje total de todas las columnas por la cantidad de columnas.

- **Heurística “Global Libre”**:
  - Temperatura inicial: 0.80 °C
  - Temperatura final: 0.00005 °C
  - Tasa de enfriamiento: 0.995
  - Puntos por coincidencia: 10.0
  - Puntos por no-coincidencia: -1.0
  - Penalidad por *gap*: -4.0
  - Cantidad máxima sin cambios de energía: 2000
  - Cantidad máxima de cambios en el estado vecino por iteración: 10
  - Estados vecinos a evaluar por iteración: 5
  - Modo: “Libre”
- **Heurística “Global Estricta”**:

- Temperatura inicial: 0.80 °C
- Temperatura final: 0.00005 °C
- Tasa de enfriamiento: 0.995
- Puntos por coincidencia: 10.0
- Puntos por no-coincidencia: -1.0
- Penalidad por *gap*: -4.0
- Cantidad máxima sin cambios de energía: 2000
- Cantidad máxima de cambios en el estado vecino por iteración: 10
- Estados vecinos a evaluar por iteración: 5
- Modo: “Estricta”

Local

Al contrario que la función interior, “local” busca ser menos estricta con la inclusión de *gaps* permitiendo que aparezcan *gaps* en las columnas internas de la matriz de secuencias alineadas.

Para evitar que la matriz de secuencias alineadas crezca indefinidamente, se divide el puntaje total de todas las columnas por la cantidad de columnas.

- **Heurística “Local Libre”**:
  - Temperatura inicial: 0.80 °C
  - Temperatura final: 0.00005 °C
  - Tasa de enfriamiento: 0.995
  - Puntos por coincidencia: 7.0
  - Puntos por no-coincidencia: 1.0
  - Penalidad por *gap*: -3.0
  - Cantidad máxima sin cambios de energía: 2000
  - Cantidad máxima de cambios en el estado vecino por iteración: 10
  - Estados vecinos a evaluar por iteración: 5
  - Modo: “Libre”

- **Heurística "Local Estricta"**:
    - Temperatura inicial: 0.80 °C
    - Temperatura final: 0.00005 °C
    - Tasa de enfriamiento: 0.995
    - Puntos por coincidencia: 7.0
    - Puntos por no-coincidencia: 1.0
    - Penalidad por *gap*: -3.0
    - Cantidad máxima sin cambios de energía: 2000
    - Cantidad máxima de cambios en el estado vecino por iteración: 10
    - Estados vecinos a evaluar por iteración: 5
    - Modo: "Estricta"

4.4. Elección de las herramientas de medición de la calidad de los MSA

Con el objeto de comparar los resultados obtenidos por cada software de MSA con las referencias curadas por los autores de la base de datos de prueba, este trabajo utiliza el paquete *MUMSA* propuesto por los autores del software KAlign. Dicha herramienta clasifica a una serie de alineamientos de acuerdo al puntaje final de calidad *"Overlap Score"* según el solapamiento con el alineamiento de referencia.

4.4.1. Sobre MUMSA

Este *software* escrito en el lenguaje C fue desarrollado por la *Swedish Graduate School for Functional Genomics and Bioinformatics* y recibe alineamientos en formato FASTA. Al ejecutarlo se debe informar el alineamiento de referencia y luego los alineamientos a comparar [63]. La herramienta realiza una evaluación comparativa de calidad de los otros archivos de alineamiento en comparación con la referencia.

El primer paso de *MUMSA* es calcular una matriz de similitud entre los pares de secuencias en la referencia y los demás archivos de alineamiento. Luego, utiliza esta matriz de similitud para estimar la calidad de cada archivo de alineamiento en comparación con la referencia.

MUMSA utiliza dos métricas de calidad al evaluar la calidad de los archivos de alineamiento: el porcentaje de identidad y el porcentaje de cobertura.

- El porcentaje de identidad se refiere a la proporción de residuos idénticos entre la referencia y el archivo de alineamiento.
- El porcentaje de cobertura se refiere a la proporción de posiciones alineadas en el archivo de alineamiento en comparación con la referencia.

Para cada archivo de alineamiento, este *software* de puntuación calcula el porcentaje de identidad y el porcentaje de cobertura en comparación con la referencia. Luego, los puntajes de calidad se normalizan y se combinan, dando lugar al puntaje final de solapamiento.

4.5. Desarrollo del flujo de trabajo

Con el fin de efectuar las pruebas de forma automatizada y reproducible se define una serie de pasos a ejecutar para cada una de las alternativas de *software* seleccionadas, más el SA desarrollado en este proyecto final. Estos pasos se implementan en el lenguaje *NextFlow* que brinda un marco de trabajo para la creación de flujos de trabajo o *workflows* [64] (figura 13).

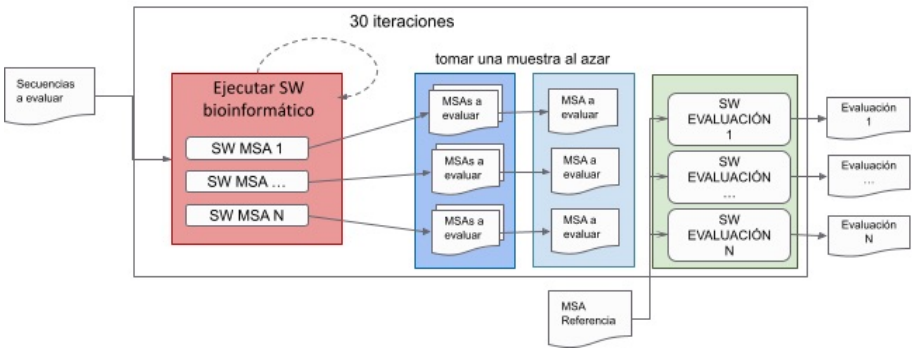

Figura 13: Esquema de los pasos a ejecutar dentro del contexto de *NextFlow*

4.5.1. Ejecución de MSASA y del *software* elegido

El *software* MSASA está implementado en el lenguaje *Python* (versión 3.12). El código fuente y su documentación correspondiente se encuentra disponible para su consulta, descarga e instalación en un repositorio de *Github* bajo la licencia *MIT*.

El código fuente incluye la documentación sobre los pasos a seguir para crear un ambiente de trabajo *Python* e instalar dentro de él las dependencias requeridas para el funcionamiento del mismo. Más detalles sobre cómo obtener el código fuente y su correcta instalación se encuentran en los anexos de este trabajo.

Cuando se utiliza un algoritmo estocástico, es necesario múltiples pruebas para obtener resultados más robustos y confiables. Esto se debe a que los algoritmos estocásticos pueden

producir diferentes soluciones en diferentes ejecuciones debido a la naturaleza estocástica de su enfoque.

Ejecutar múltiples pruebas proporciona una forma de evaluar la variabilidad de los resultados y obtener una estimación más precisa del desempeño promedio del algoritmo. Al realizar varias ejecuciones, se puede obtener una medida de la dispersión de los resultados y evaluar la consistencia del algoritmo en diferentes instancias del problema. En general, ejecutar 30 veces un escenario de prueba es comúnmente aceptado para obtener una buena estimación de la variabilidad y el rendimiento promedio del algoritmo.

Una vez definido este marco de trabajo para cada herramienta, se generan los MSA de los 23 casos de prueba (enumerados en la tabla 23), al ejecutar en forma paralela y automática, los siguientes *software* MSA:

1. Clustal Omega
2. KAlign
3. MAFFT
4. Muscle
5. T-Coffee
6. MSASA hibridado con “Identidad Libre”
7. MSASA hibridado con “Identidad Estricta”
8. MSASA hibridado con “Coincidencias Libre”
9. MSASA hibridado con “Coincidencias Estricta”
10. MSASA hibridado con “Similitud Blosum62 Libre”
11. MSASA hibridado con “Similitud Blosum62 Estricta”
12. MSASA hibridado con “Similitud PAM250 Libre”
13. MSASA hibridado con “Similitud PAM250 Estricta”
14. MSASA hibridado con “Similitud Gonnet92 Libre”
15. MSASA hibridado con “Similitud Gonnet92 Estricta”

16. MSASA hibridado con “Global Libre”
17. MSASA hibridado con “Global Estricta”
18. MSASA hibridado con “Local Libre”
19. MSASA hibridado con “Local Estricta”

Mediante el operador `take` de *NextFlow* se puede extraer un resultado de MSA al azar entre los 30 resultados obtenidos de cada alternativa de *software* MSA evaluada para proseguir con el estudio de las métricas de calidad utilizando el paquete *MUMSA*.

5. Análisis de los resultados

A continuación se describen los datos obtenidos al ejecutar cada *software* de evaluación sobre las muestras de MSA obtenidas. Estos resultados fueron agrupados por la métrica estudiada y las dimensiones utilizadas para separar los casos de pruebas.

Las interpretaciones presentadas en este capítulo sientan las bases de las conclusiones expresadas en el capítulo siguiente. En relación a la metodología de utilizada en esta etapa, los resultados de cada métrica son cargados en una hoja de cálculo y posteriormente se extraen medidas estadísticas y gráficos desde tablas dinámicas usando librerías de *Python* como *Pandas* y *MatPlotLib*.

5.1. Estudio de calidad utilizando la métrica *Overlap Score* de *MUMSA*

A primera vista, los resultados muestran que los *software* del estado del arte están preparados para generar alineamientos cercanos a las referencias establecidas por los curadores de BAli BASE. Se observa una amplia diferencia al comparar los valores de mediana (tabla 24) y promedio (tabla 25)

Longitud de las secuencias	Cantidad de secuencias	MSASA	Otros
CL	AC	0.109019	0.800397
	BC	0.523124	0.905678
LL	AC	0.068786	0.733084
	BC	0.037896	0.740336
ML	AC	0.323811	0.919941
	BC	0.148823	0.804148

Tabla 24: Media comparada entre los diferentes grupos de estudio

Longitud de las secuencias	Cantidad de secuencias	MSASA	Otros
CL	AC	0.134	0.789
	BC	0.489286	0.892975
LL	AC	0.074780	0.717893
	BC	0.083606	0.686073
ML	AC	0.354017	0.922930
	BC	0.202845	0.716670

Tabla 25: Promedio comparado entre los diferentes grupos de estudio

5.1.1. Perspectiva general

No obstante, es importante recalcar que MSASA ha podido generar para ciertos escenarios de prueba, resultados relativamente cercanos a los que se obtienen con las otras herramientas (figura 14):

- **CL-BC**: Para cortas longitudes y baja cantidad de secuencias, la media es de 52 %
- **ML-AC**: Para medianas longitudes y alta cantidad de secuencias, la media es de 0.32 %

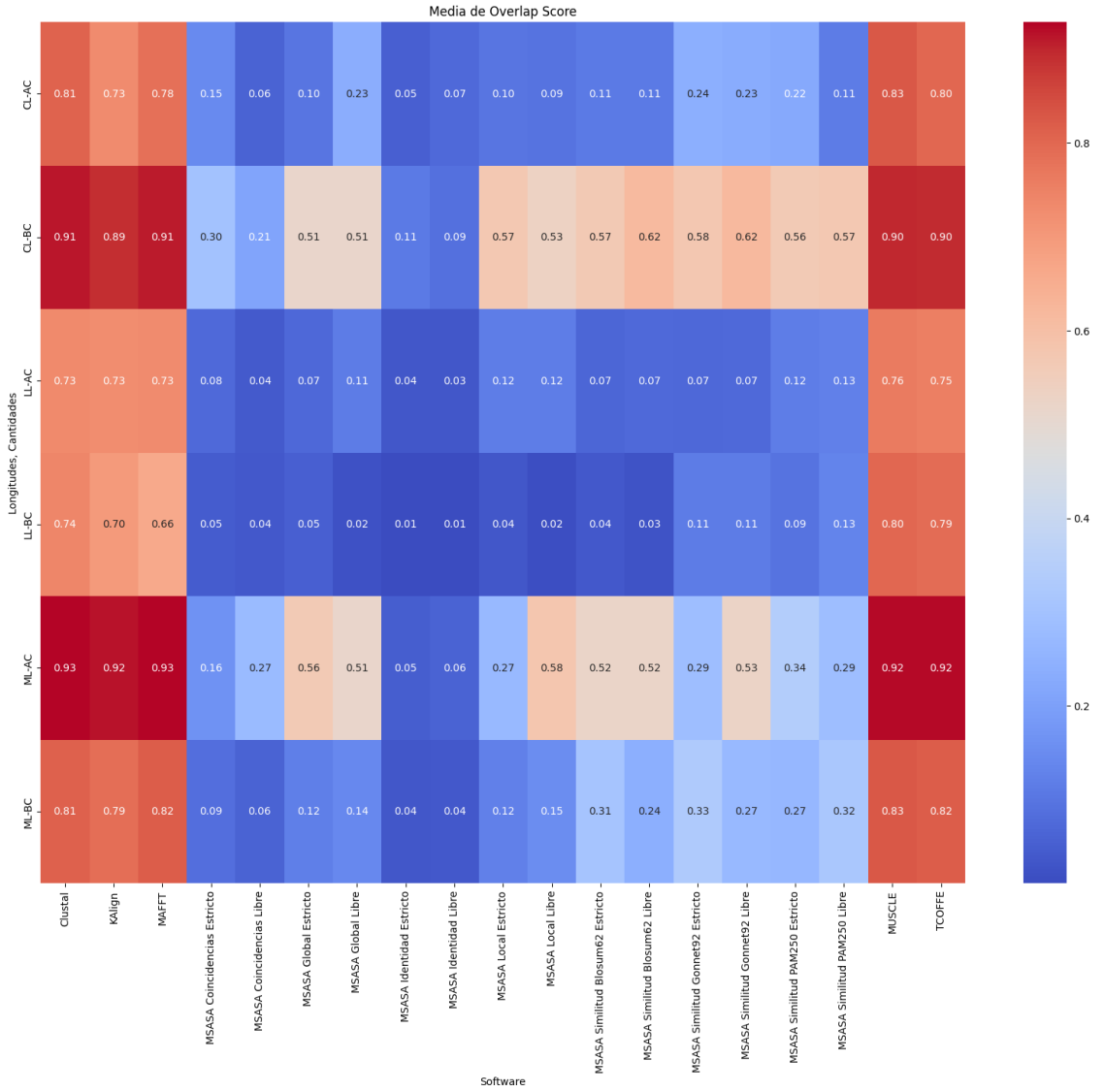

Figura 14: Mapa de calor donde se visualiza la media de la métrica estudiada en cada *software* MSA para los diferentes grupos en base a la cantidad de secuencias y longitud de las mismas. Mientras más oscuro el tono de rojo, más cercano al alineamiento de referencia. Al contrarío, mientras más oscuro sea el tono azul, más lejos de la referencia.

Al consolidar los resultados de todas las funciones hibridadas en MSASA y compararlas contra todos los *software* del estado del arte, también se observa que hay ciertos grupos que

tienen resultados interesantes para estudiar (figura 15). El grupo “CL-BC”, es decir, *Cortas longitudes* y *Baja cantidad de secuencias* tiene una media superior al 0.50 de *Overlap Score* con respecto a las referencias.

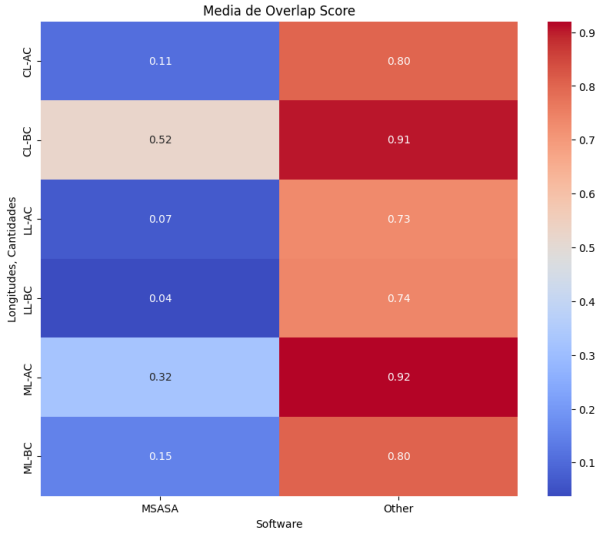

Figura 15: Mapa de calor donde se visualiza la media de la métrica estudiada en cada *software* MSA para los diferentes grupos en base a la cantidad de secuencias y longitud de las mismas. Mientras más oscuro el tono de rojo, más cercano al alineamiento de referencia. Al contrarío, mientras más oscuro sea el tono azul, más lejos de la referencia.

Tanto al analizar los resultados por grupos de cantidad de secuencias (figura 16) como longitud de las mismas (figura 17), se observa un patrón: las heurísticas hibridadas con matrices de substitución son las que producen alineamientos más cercanos a las referencias. No hay un patrón definido entre los modos libres y estrictos.

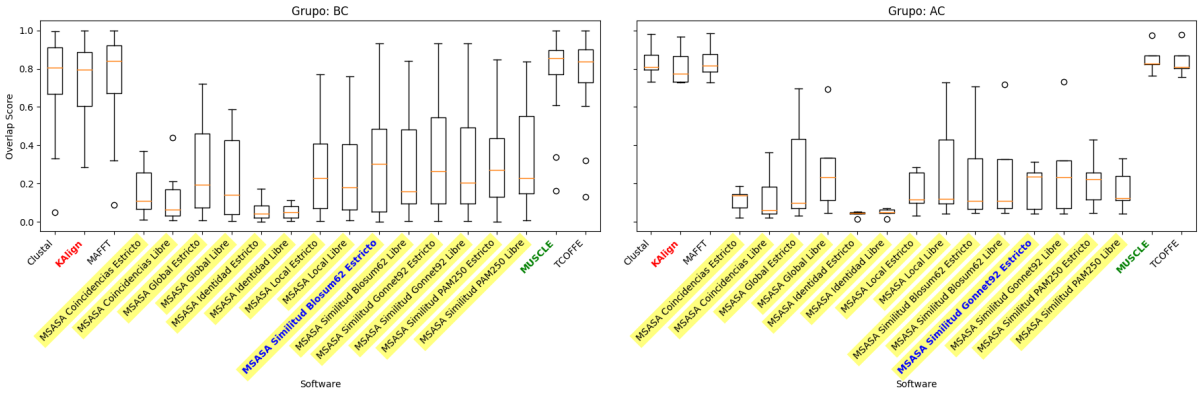

Figura 16: Comparación de la métrica provista por MUMSA agrupando por grupos de cantidad de secuencias: “BC” para *Bajas cantidades* y “AC” para *Altas cantidades*. En rojo se señala el *software* del estado del arte con menor superposición en promedio, mientras que en verde se señala al mejor. En amarillo se marcan las heurísticas hibridadas con MSASA y en azul la que más se acerca al alineamiento de referencia provisto por *BAli BASE*.

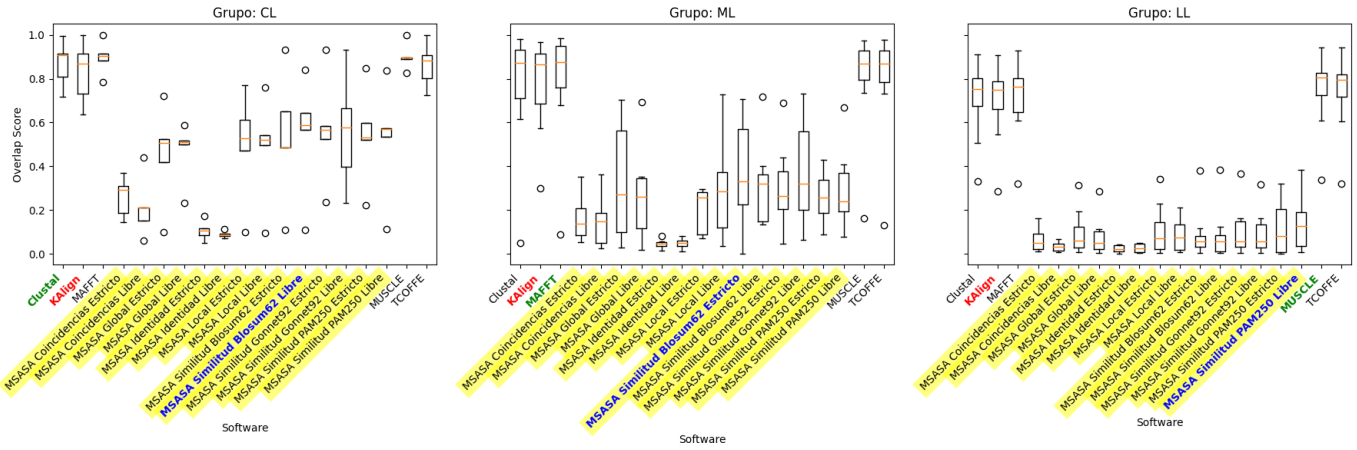

Figura 17: Similar a la gráfica anterior, pero agrupando por grupos de longitud de las secuencias: “CL” para *Cortas longitudes*, “ML” para *Medianas longitudes* y “LL” para *Largas longitudes*.

Se desprende de este análisis que las matrices de substitución contienen mayor información biológica sobre la conservación de regiones dentro alineamientos de secuencias que las funciones heurísticas creadas en este trabajo (tabla 26). Esto tiene sentido dado que las tres matrices han sido fruto del trabajo de investigación de la evolución y las propiedades químicas de los residuos que forman parte de las cadenas de péptidos. De cierta manera, MSASA demuestra la importancia de esa información biológica (bioquímica, evolutiva) para el estudio bioinformático de entidades biológicas como proteínas o cadenas de ADN/ARN.

Software	**Promedio de *Overlap Score***
MUSCLE	0.781546
TCOFFE	0.768422
MAFFT	0.762926
Clustal	0.745632
KAlign	0.737890
MSASA Similitud Gonnet92 Libre	0.296055
MSASA Similitud Blosum62 Estricto	0.282559
MSASA Similitud Gonnet92 Estricto	0.281984
MSASA Similitud PAM250 Libre	0.278591
MSASA Similitud PAM250 Estricto	0.267664
MSASA Similitud Blosum62 Libre	0.267332
MSASA Global Estricto	0.245984
MSASA Local Libre	0.237284
MSASA Global Libre	0.223812
MSASA Local Estricto	0.218461
MSASA Coincidencias Estricto	0.134879
MSASA Coincidencias Libre	0.108595
MSASA Identidad Estricto	0.047792
MSASA Identidad Libre	0.046518

Tabla 26: Promedio de la métrica provista por *MUMSA* al agrupar por *software*.

A continuación de las técnicas basadas en las matrices se encuentran las heurísticas de Global y Local que intentan replicar el alineamiento de dos secuencias. Las cuatro alternativas se encuentran muy cerca entra (22.38 % y 24.59 %). Esto puede entenderse debido a que son técnicas rudimentarias que buscan aplicar los conceptos creados por bioinformáticos en el pasado para alineamientos globales y locales (*Needleman-Wunsch* y *Smith-Waterman*).

Finalmente, encontramos los métodos más triviales: Identidad y Coincidencias. Los dos son intentos computacionalmente sencillos y agnósticos de la realidad biológica. Cómo es de esperar, en promedio, proveen los alineamientos más lejanos a las referencias (de un 4.6 % a un 13 % de *solapamiento*).

5.1.2. Casos de estudio específicos

Esta sección muestra en detalle los resultados para ciertos casos de estudio: C1, C2, C14, y C22 para remarcar la calidad de las heurísticas hibridadas en MSASA.

Caso de estudio C1 - BBS12014

Este es un ejemplo (figura 18) que muestra el potencial de MSASA para alinear secuencias cortas (CL) en bajas cantidades (BC). Si bien los resultados de los *software* del estado del arte logran el mismo resultado que la referencia, los alineamientos de MSASA que utilizan conceptos biológicos obtienen resultados muy buenos:

- **Heurísticas con matrices de substitución**: Obtienen solapamientos entre 84 % y 93 %.
- **Heurísticas Global/Local**: Entre 59 % y 77 %.
- **Heurísticas con triviales**: Incluso la función de coincidencias logra solapar un 31 % de las columnas.

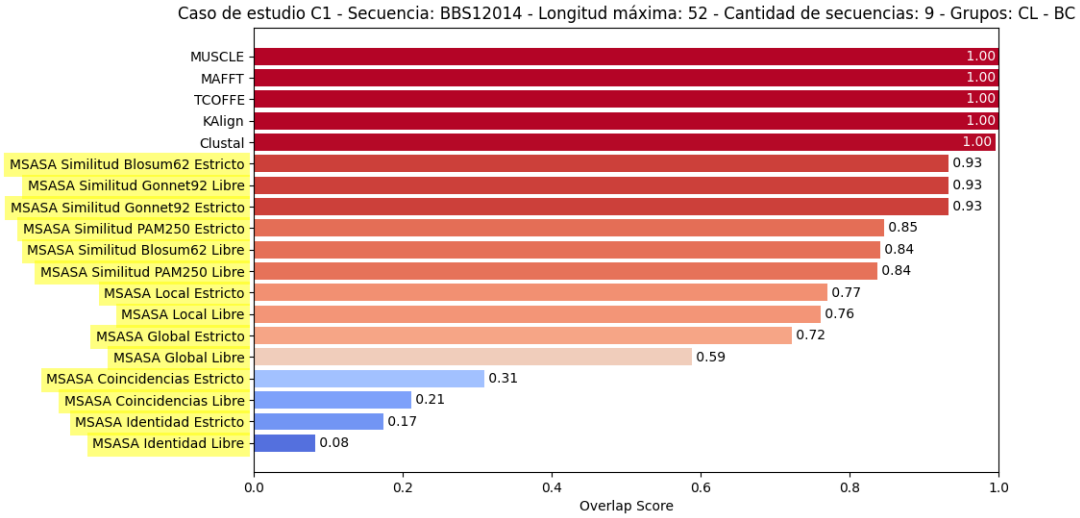

Figura 18: Comparación del *Overlap Score* por cada *software* ejecutado para el caso de estudio C1.

A través del estudio del alineamiento de referencia y el mejor generado por MSASA utilizando el *software* JalView [65], se observa al comparar las cadenas consenso de la referencia (figura 19) que algunos posiciones son iguales en la consenso extraída de Similitud Blosum62 estricta (figura 20). La heurística es capaz de capturar parte de la información evolutiva y de conservación de las secuencias.

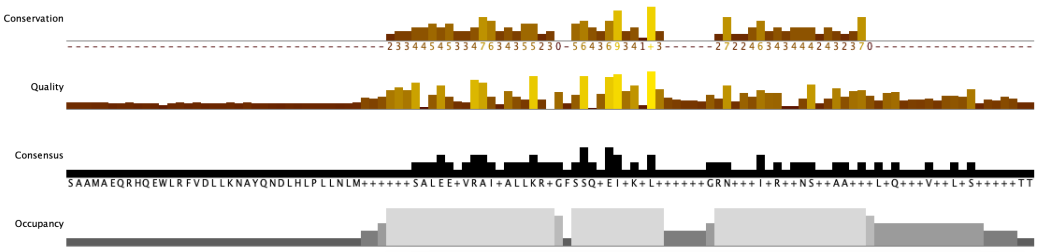

Figura 19: Análisis realizado mediante JalView para estudiar las características del alineamiento de referencia para el caso de estudio C1 - BB11013.

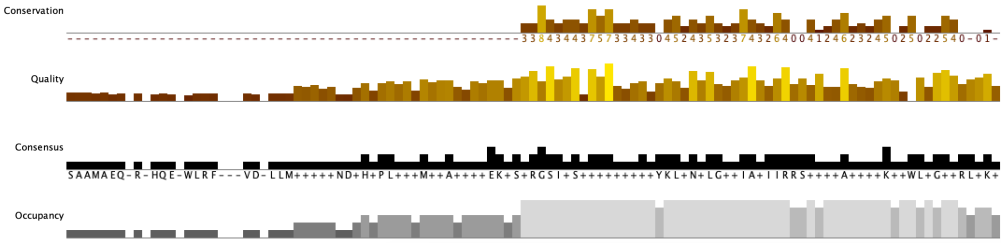

Figura 20: Mismo análisis en JalView pero utilizando el alineamiento Similitud Blosum62 Estricta para el caso de estudio C1 - BB11013.

Caso de estudio C2 - BB11013

Aquí se observa (figura 22) que el caso C2 es un caso difícil para todos los *software* utilizado, tanto los del estado del arte como MSASA con sus funciones hibridadas. Es remarcable que las heurísticas de Similitud e incluso Local están puntuados entre herramientas MSA del estado de arte.

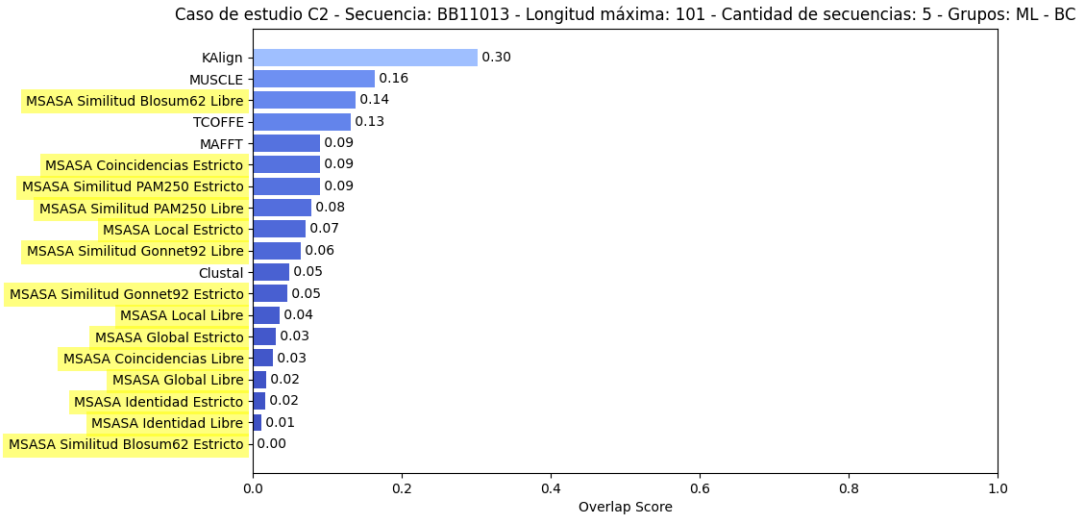

Figura 21: Comparación del *Overlap Score* por cada *software* ejecutado para el caso de estudio C2.

Caso de estudio C14 - BBS50005

Finalmente, un ejemplo donde se observa como el software del estado del arte alcanza valores suficientemente buenos al comparar con la referencia, mientras que MSASA no supera el 20 % de solapamiento. Si bien es un porcentaje muy bajo, sirve para demostrar como las heurísticas triviales dan los resultados más alejados de la referencia y a continuación, las otras funciones van acercándose sucesivamente al 20 %.

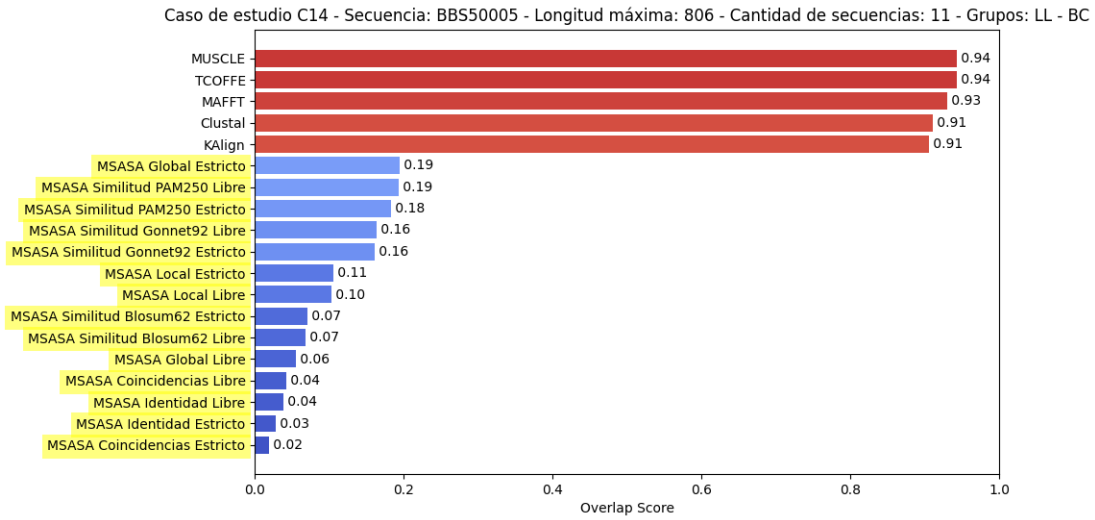

Figura 22: Comparación del *Overlap Score* por cada *software* ejecutado para el caso de estudio C2.

6. Conclusiones y debate

Una vez finalizada la etapa de discusión de los resultados, se presentan en esta sección las conclusiones organizadas en tres niveles: el primero sobre el presente de la bioinformática y cómo el conocimiento extraído a partir de los datos generados hace que se creen nuevas herramientas y métodos. De esta manera se quiere demostrar la importancia de conceptos como el MSA o las secuencias consenso. Luego se plantean las dificultades que SA tiene al resolver el problema de MSA en comparación con las otras herramientas actuales. Este capítulo cierra con unas notas a futuro sobre cómo este trabajo puede aportar un marco de trabajo para futuras investigaciones en el desarrollo de mejores aplicaciones de MSA.

6.1. Conclusiones

6.1.1. Sobre la implementación SA en el problema de MSA

Luego de implementar MSASA con sus variantes heurísticas y compararlo con las herramientas MSA disponibles actualmente, queda claro que la calidad de los alineamientos en general no está al mismo nivel que el estado del arte. Sin embargo, en algunos grupos de estudio, los resultados usando heurísticas apoyadas en conceptos biológicos pueden ser comparables con los de algunas herramientas. El capítulo anterior presenta los detalles de los resultados que ayudan a entender en que escenarios MSASA puede ser una alternativa a estudiar y mejorar. Nótese que los resultados obtenidos no han necesitado de estructuras complejas, como ser árboles guías, sino que mediante estudiando las columnas del alineamiento se pueden obtener resultados relativamente buenos.

Este trabajo práctico final sirve para validar la importancia de conectar los conceptos biológicas en los algoritmos: al incorporar el uso de las matrices de conservación de aminoácidos se encuentran resultados mejores en forma notable. En la misma dirección, los intentos de replicar los algoritmos de alineamiento de pares globales y locales son el segundo grupo de heurísticas con mejores resultados.

A futuro, también puede evaluarse la compilación del código al lenguaje C o utilizar técnicas para ejecutar código en GPU para acelerar los tiempos de ejecución. Una de las desventajas de analizar combinaciones de elementos columna por columna, reside en la cantidad de evaluaciones a realizar que crece exponencialmente. En este trabajo se utiliza una limitación de 20 horas máximo por predicción y nunca fue alcanzado.

6.1.2. Sobre las hipótesis planteadas

El objetivo global de este trabajo abarca analizar las siguientes hipótesis:

"Dado que encontrar un MSA es un problema que computacionalmente no puede resolverse en un tiempo polimonial, surgen las siguientes hipótesis:"

1. **H1**: Los algoritmos aproximados, como las metaheurísticas, basados en SA resuelven eficazmente el MSA.

2. **H2**: Estos algoritmos (SA) son competitivos con el estado del arte a la hora de resolver el MSA.

3. **H3**: Es posible combinar o hibridar diferentes funciones objetivo para encontrar un MSA suficientemente bueno en comparación con las otras alternativas en el estado del arte.

4. **H4**: Se puede crear un marco de trabajo para comparar diferentes piezas de *software* MSA usando métricas del estado del arte para ordenar por calidad de los resultados las herramientas.

Sobre la H1

Mediante la evaluación de los resultados comprueba que las herramientas de *software* existentes resuelven el problema informático de MSA en forma satisfactoria tanto a nivel de tiempos de ejecución como en calidad de los resultados. Esto se puede validar observando como los MSA obtenidos se acercan a los valores diseñados por los expertos que curaron los alineamientos de prueba de la base de secuencias *BAli-Base*.

Sobre la H2

Es cierto para ciertos escenarios donde los resultados son similares a los obtenidos por las herramientas del estado de arte.

Sobre la H3

El algoritmo MSASA provisto en este trabajo ofrece un método de trabajo lo suficientemente flexible para incorporar nuevas funciones que puedan aumentar la calidad de los alineamientos. Como se dice en este capítulo, las heurísticas de Similitud con las matrices de substitución dan resultados muy promisorios.

Sobre la H4

El marco de trabajo es lo suficientemente flexible para ser adaptado tanto para incorporar nuevas piezas de *software* MSA como métricas de calidad a fin de evaluar los resultados de alineamientos múltiples.

6.2. Debate

Encontrar información a partir de datos bioinformáticos es una tarea cada vez más crucial en las ciencias asociadas a la biología. Los alineamientos múltiples de secuencias, secuencias consenso en otra información son la base para los próximos avances bioinformáticos. El problema de la obtención de datos de entidades biológicas como genes, proteínas, fragmentos de ADN/ARN parece estar resuelto gracias a los avances de *hardware* y *software*. Es el momento de generar valor partiendo de toda los archivos almacenados en miles de bases de datos para que nuevas investigaciones puedan llevarse a cabo. Por ejemplo, *AlphaFold* irrumpió a fines de la década de 2010s con su método disruptivo de predicción de plegamiento y estructura tridimensional de las proteínas. Su éxito en gran medida, más allá de sus avances en el uso de inteligencia artificial, se debe a la importancia de la conservación de secuencias. El componente evolutivo requerido por la inteligencia artificial es un MSA a partir de secuencias similares a la proteína a plegar.

6.2.1. SA como base para otras funciones a optimizar

Además de las funciones heurísticas implementadas, evaluadas y comparadas anteriormente, el código MSASA de este trabajo práctico es lo suficientemente versátil como para que nuevas funciones objetivo puedan ser incorporadas y de tal manera, se efectúe una optimización al minimizar la energía del sistema.

A futuro pueden agregarse nuevas funciones que agreguen más contexto al cálculo del puntaje de cada alineamiento. Una alternativa es considerar más de una columna al momento de obtener el puntaje por columna. Agregando la información de la columna anterior y la siguiente, se abre el juego a nuevas condiciones de puntaje. Por ejemplo, podrían distinguirse nuevos *gaps*, o *gaps* de cierre.

Incluso, en lugar de estudiar las columnas de la matriz del alineamiento, se puede adaptar el código para estudiar las filas. De esta forma, pueden haber funciones heurísticas que crean combinaciones de filas y por cada una, efectúa un alineamiento global o local para obtener un

puntaje.

6.2.2. Marco de trabajo

Independientemente de los resultados obtenidos, este trabajo provee un marco de trabajo de futuras investigaciones que busquen implementar nuevas ideas biológicas dentro de un modelo computacional aplicado al problema de MSA. El proyecto está desarrollado para que sea lo más genérico posible de forma tal que sean incorporadas nuevas funciones de energía del sistema sencillamente. Véase el caso de TCS como base para futuras funciones objetivos.

También esta tesis implementa una forma estructurada de ejecutar comparaciones entre distintos piezas de *software* de MSA al definir flujos de trabajo con la herramienta *NextFlow*. Todo lo programado y desarrollado a lo largo de este proyecto está disponible en el repositorio público en *GitHub*. Además, en los anexos de este trabajo el lector puede encontrar las decisiones de diseño e implementación del MSASA como los flujos de ejecución y evaluación pertinentes.

Bibliografía

[1] Barreto-Hernández E. «Bioinformática: Historia y perspectivas futuras». En: *Rev. Colombia Ciencia y Tecnología* 20(3) (2002), págs. 36-44.

[2] Cañedo Andalia R y Arencibia Jorge R. «Bioinformática: en busca de los secretos moleculares de la vida». es. En: *ACIMED* 12 (dic. de 2004), págs. 1-1. ISSN: 1024-9435. URL: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352004000600002&nrm=iso.

[3] Casillas Viladerrams S. *Development and application of bioinformatic tools for the representation and analysis of genetic diversity*. Universitat Autònoma de Barcelona, 2008.

[4] Wang L y Jiang T. «On the Complexity of Multiple Sequence Alignment». En: *Journal of computational biology : a journal of computational molecular cell biology* 1 4 (1994), págs. 337-48.

[5] Kirkpatrick S, Gelatt Jr CD y Vecchi MP. «Optimization by simulated annealing». En: *Science* 220 (1983), págs. 671-680.

[6] Patrias K y Wendling D. «MedlinePlus en español [Internet]». En: *"Bethesda (MD): National Library of Medicine (US)"* (2019).

[7] Watshon J y Crick F. «Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid». En: *Nature* 171 (1953), págs. 737-738. DOI: https://doi.org/10.1038/171737a0.

[8] Pray L A. «Discovery of DNA Structure and Function: Watson and Crick». En: *Nature Education* 1 (2008).

[9] Crick F. «Central Dogma of Molecular Biology». En: *Nature* 227 (1970), págs. 561-563. DOI: https://doi.org/10.1038/227561a0.

[10] Chaffey N. «Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K. and Walter, P. Molecular biology of the cell. 4th edn.» En: *Annals of Botany* 91 (feb. de 2003), págs. 401-401. DOI: 10.1093/aob/mcg023.

[11] Cox M y Nelson D. *Lehninger Principles of Biochemistry*. Vol. 5. Ene. de 2000. DOI: 10.1007/978-3-662-08289-8.

[12] Karp G. «Expresión génica: de la transcripción a la traducción». En: *Biología celular y molecular. Conceptos y experimentos, 7e*. New York, NY: McGraw-Hill Education, 2017. URL: accessmedicina.mhmedical.com/content.aspx?aid=1139754179.

[13] Bartlett JMS y Stirling D. «A Short History of the Polymerase Chain Reaction». En: *Methods in Molecular Biology* 226 (2003). DOI: https://doi.org/10.1385/1-59259-384-4:3.

[14] Sanger F, Nicklen S y Coulson AR. «DNA sequencing with chain-terminating inhibitors». En: *Proc Natl Acad Sci U S A* 74(12) (1977), págs. 5463-5467. DOI: 10.1073/pnas.74.12.5463.

[15] López de Heredia U. «Las técnicas de secuenciación masiva en el estudio de la diversidad biológica». En: *Munibe Ciencias Naturales* 64 (2016), págs. 7-31. DOI: 10.21630/mcn.2016.64.07.

[16] Moore GE. «Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.,» en: *IEEE Solid-State Circuits Society Newsletter* 11 no. 3 (2006), págs. 33-35. DOI: 10.1109/N-SSC.2006.4785860.

[17] Hernández M et al. «Aplicación de la secuenciación masiva y la bioinformática al diagnóstico microbiológico clínico». En: *Revista Argentina de Microbiología* 52.2 (), págs. 150-161. ISSN: 0325-7541. DOI: https://doi.org/10.1016/j.ram.2019.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0325754119300811.

[18] Schneider TD y Stephens RM. «Sequence Logos: A New Way to Display Consensus Sequences». En: *Nucleic Acids Res* 18 (1990), págs. 6097-6100.

[19] Notredame C, Higgins DG y Heringa J. «T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment». En: *J. Mol. Biol.* 302 (2000), págs. 205-217. DOI: doi:10.1006/jmbi.2000.4042.

[20] Edgar RC. «MUSCLE: multiple sequence alignment with high accuracy and high throughput». En: *Nucleic Acids Research* 32.5 (2004), págs. 1792-1797.

[21] Aluru S. *Handbook of Computational Molecular Biology (Chapman & All/Crc Computer and Information Science Series)*. Chapman Hall/CRC, 2005. ISBN: 1584884061.

[22] Chiner-Oms A y González-Candelas F. «EvalMSA: A Program to Evaluate Multiple Sequence Alignments and Detect Outliers». En: *Evolutionary Bioinformatics* 12 (2016), EBO.S40583. DOI: 10.4137/EBO.S40583.

[23] Ahola V et al. «A statistical score for assessing the quality of multiple sequence alignments». En: *BMC Bioinformatics* 7 (2006). DOI: 10.1186/1471-2105-7-484.

[24] Bawono P y Heringa J. *Comprehensive Biomedical Physics*. Vol. 6. 2014. ISBN: 978-0-444-53633-4.

[25] Strasser B. «Collecting, Comparing, and Computing Sequences: The Making of Margaret O. Dayhoff's Atlas of Protein Sequence and Structure, 1954-1965». En: *Journal of the history of biology* 43 (dic. de 2010), págs. 623-60. DOI: 10.1007/s10739-009-9221-0.

[26] Henikoff S y Henikoff JG. «Amino acid substitution matrices from protein blocks.» En: *Proceedings of the National Academy of Sciences* 89.22 (1992), págs. 10915-10919. DOI: 10.1073/pnas.89.22.10915. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.89.22.10915. URL: https://www.pnas.org/doi/abs/10.1073/pnas.89.22.10915.

[27] Bioinfo Perl. *Ideas y código para problemas de genómica de plantas, biología computacional y estructural*. URL: https://bioinfoperl.blogspot.com/2012/07/matrices-de-sustitucion-y-alineamiento.html. (accessed: 04.07.2012).

[28] Gonnet G, Cohen M y Benner S. «Exhaustive Matching of the Entire Protein Sequence Database». En: *Science* 256.5062 (1992), págs. 1443-1445. DOI: 10.1126/science.1604319. eprint: https://www.science.org/doi/pdf/10.1126/science.1604319. URL: https://www.science.org/doi/abs/10.1126/science.1604319.

[29] Cortéz A. «Teoría de la complejidad computacional y teoría de la computabilidad». En: *Revista de investigación en sistemas de información, Facultad de Ingenería de Sistemas e Informática, Univeridad Nacional Mayor de San Marcos* 1 (2004), págs. 102-105. ISSN: 1815-0268.

[30] Turing AM. «On Computable Numbers, with an Application to the Entscheidungsproblem». En: *Proceedings of the London Mathematical Society* 2 (42 1937), págs. 230-265.

[31] Copeland B. «The Church-Turing Thesis». En: *The Stanford Encyclopedia of Philosophy (Summer 2020 Edition)* (). Ed. por Edward N. Zalta. URL: https://plato.stanford.edu/archives/sum2020/entries/church-turing/.

[32] Needleman SB y Wunsch CD. «A general method applicable to the search for similarities in the amino acid sequence of two proteins». En: *Journal of Molecular Biology* 48 (3 1970), págs. 443-453.

[33] Gotoh O. «An improved algorithm for matching biological sequences». En: *Journal of Molecular Biology* 162.3 (1982), págs. 705-708. ISSN: 0022-2836. DOI: https://doi.org/10.1016/0022-2836(82)90398-9. URL: https://www.sciencedirect.com/science/article/pii/0022283682903989.

[34] Smith TF y Waterman MS. «Identification of Common Molecular Subsequences». En: 147 (1) (1981), págs. 195-197. DOI: doi:10.1016/0022-2836(81)90087-5.

[35] Farrar M. «Striped Smith–Waterman speeds database searches six times over other SIMD implementations». En: *Bioinformatics* 23.2 (nov. de 2006), págs. 156-161. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl582. URL: https://doi.org/10.1093/bioinformatics/btl582.

[36] Eddy SR. «Profile hidden Markov Models». En: *Bioinformatics* 14 (1998), págs. 755-763.

[37] Notredame C e Higgins DG. «SAGA: Sequence Alignment by Genetic Algorithm». En: *Nucleic Acids Research* 24 (8 1996), págs. 1515-1524.

[38] Kim J, Pramanik S y Chung MJ. «Multiple sequence alignment using simulated annealing». En: *Comput Appl Biosci* 10 (4 1994), págs. 419-26. DOI: 10.1093/bioinformatics/10.4.419.

[39] Higgins DG y Sharp PM. «CLUSTAL: a package for performing multiple sequence alignment on a microcomputer.» En: *Gene* 73 1 (1988), págs. 237-44.

[40] Feng DF y Doolittle RF. «Progressive sequence alignment as a prerequisitetto correct phylogenetic trees». En: *Journal of Molecular Evolution* 25 (1987), págs. 351-360.

[41] Baum LE et al. «A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains». En: *Annals of Mathematical Statistics* 41 (1970), págs. 164-171.

[42] Eddy SR. «Profile hidden Markov models». En: *Bioinformatics* 14 9 (1998), págs. 755-63.

[43] Sievers F e Higgins DG. «Clustal Omega for making accurate alignments of many protein sequences». En: *Protein Sci* 27 (2018), págs. 135-145.

[44] Edgar RC. «MUSCLE v5 enables improved estimates of phylogenetic tree confidence by ensemble bootstrapping». En: *bioRxiv* (2021). DOI: https://doi.org/10.1101/2021.06.20.449169.

[45] Lassmann T, Frings O y Sonnhammer ELL. «Kalign2: high-performance multiple alignment of protein and nucleotide sequences allowing external features». En: *Nucleic Acids Research* 37 (2009), págs. 858-865.

[46] Katoh K et al. «MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform». En: *Nucleic Acids Res.* 30 (2002), págs. 3059-3066.

[47] Melián B, Pérez JA y et al. «Metaheurísticas: una visión global». En: *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* 19 (2003), págs. 7-28. ISSN: 1137-3601.

[48] Puris A, Novoa P y Oviedo B. «Desarrollo de metaheurísticas poblacionales para la solución de problemas complejos». En: *Editorial Compás, Guayaquil Ecuador* (2020), pág. 136.

[49] Polya G. «How to Solve It». En: *Princeton University Press* (1957).

[50] Wolpert DH y Macready WG. «No Free Lunch Theorems for Optimization». En: *IEEE Transactions of Evolutionary Computation* 1 (1997), págs. 67-82.

[51] Osman IH y Kelly JP. *MetaHeuristics: Theory and Applications*. Kluwer Academic, 1996.

[52] Glover F y Kochenberger G. «Handbook of Metaheuristics». En: *Kluwer Academic Publishers* (2003).

[53] Martí R. «Procedimientos Metaheurísticos en Optimización Combinatoria». En: *Revista Virtual Pro, Investigación de operaciones* 5 (2014). Ed. por Univesitat de Valencia.

[54] Bertsimas D y Tsitsiklis J. «Simulated Annealing». En: *Statistical Science* 8 (1 1993), págs. 10-15.

[55] Holland JH. «Genetic algorithms». En: *Scientific American* (1992).

[56] Engelbrecht AP. «Fundamentals of Computational Swarm Intelligence». En: (2006).

[57] Lee YS et al. «A comparison of the performance of multi-objective optimization methodologies for solvent design». En: *29th European Symposium on Computer Aided Process Engineering*. Ed. por Kiss AA et al. Vol. 46. Computer Aided Chemical Engineering. Elsevier, 2019, págs. 37-42. DOI: https://doi.org/10.1016/B978-0-12-818634-3.50007-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128186343500072.

[58] Černý V. «Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm». En: *J Optim Theory Appl* (1985). DOI: https://doi.org/10.1007/BF00940812.

[59] Metropolis N et al. «Equation of State Calculations by Fast Computing Machines». En: *The Journal of Chemical Physics* 21.6 (jun. de 1953), págs. 1087-1092. DOI: 10.1063/1.1699114. URL: https://doi.org/10.1063/1.1699114.

[60] Alfonso H et al. «Metaheurísticas aplicadas a problemas de optimización». En: (2010). DOI: http://oai:sedici.unlp.edu.ar:10915/19440.

[61] "Bahr A et al. «BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations». En: *Nucleic Acids Res.* 29 (2001), "323-326". DOI: "https://doi.org/10.1093/nar/29.1.323".

[62] Thompson JD y et al. «A comprehensive comparison of multiple sequence alignment programs». En: *Nucleic acids research* 27 (1999), págs. 2682-90. DOI: "doi:10.1093/nar/27.13.2682".

[63] Lassmann T y Sonnhammer ELL. «Kalign, Kalignvu and Mumsa: web servers for multiple sequence alignment». En: *Nucleic Acids Research* 34 (2006), W596-W599.

[64] Di Tommaso P et al. «Nextflow enables reproducible computational workflows». En: *Nature Biotechnology* 35 (abr. de 2017), págs. 316-319. DOI: 10.1038/nbt.3820.

[65] Waterhouse AM et al. «Jalview Version 2 - A multiple sequence alignment editor and analysis workbench». En: *Bioinformatics* 25 (2009). DOI: doi:10.1093/bioinformatics/btp033.

A. Anexos

A.1. Obtener una copia del código fuente desde el repositorio

De pre-requisitos, la computadora donde se instala esta pieza de *software* cuenta con:

- ***Git***: el paquete informático para usar el protocolo Git de gestión de código fuente.
- ***Python* v3.12**: El lenguaje de programación interpretado en su versión 3.
- ***Conda***: Administrador de librerías y ambientes de trabajo de *Python*.

Los pasos para descargar la última versión del código fuente desde *Github* son los siguientes:

1. Desde una sesión de consola o terminal, se debe clonar el contenido del repositorio al destino elegido: `git clone https://github.com/agdiaz/msasa_2024.git`
2. Acceder al nuevo directorio creado: `cd msasa_2024`

Las dependencias necesarias para usar este *software* están listadas en el archivo `enviroment.yaml`:

- **NumPy**: Librería con licencia BSD desarrollada para el manejo de vectores/arreglos y matrices además de una gran cantidad de funciones matemáticas para operar con arreglos vectoriales y matriciales.
- **Pandas**: Librería con licencia New-BSD para la manipulación de datos en forma de tablas numéricas y series temporales.
- **BioPython**: Librería con licencia Open-Source propia que contiene una colección de funciones y herramientas bioinformáticas.
- **Matplotlib**: Librería bjo licencia propia (licencia Matplotlib) usada para generar gráficos en Python.
- ***NextFlow***: El *software* necesario para ejecutar las pruebas y analizar los las métricas de calidad.

A. Acceso al material de la defensa pública

La defensa pública realizada en forma virtual el día 25 de octubre de 2024 puede reproducirse desde la plataforma YouTube a través del enlace: `https://www.youtube.com/live/3P_dumuGX7A?si=jKfMkvK5Xzepl-Np`.

Las diapositivas presentadas se encuentran disponibles dentro del repositorio de código fuente del proyecto: `https://github.com/agdiaz/msasa_2024/blob/main/public_defence_slides.pdf`.

B. Notas de la defensa

Durante la defensa pública se registraron las siguientes notas durante el debate con los jurados;

- Las heurísticas fueron probadas en este trabajo en forma independiente en todos los escenarios de prueba. Un comentario muy relevante es que se pueden intentar combinar heurísticas complementarias para intentar obtener mejores valores. A su vez, se propone ajustar los casos de prueba para cada función objetivo. Por ejemplo, las funciones basadas en heurísticas que emulan alineadores globales deberían probarse solamente con casos de prueba de alta similitud. De esta forma, se optimizaría el tiempo de pruebas y los resultados serían más atinados biológicamente hablando.
- Si bien la metodología MSASA está pensada para crear alineamientos múltiples a partir de secuencias, se plantea la idea de utilizar la implementación a partir de un MSA previamente generado por otro *software* de forma tal que MSASA sea un usado como un ajustador/refinador a fin de mejorar el alineamiento de entrada.
- En esta misma línea, se sugiere realizar pruebas para validar si MSASA puede ser un buen refinador combinando diferentes heurísticas.

C. Agradecimientos

Al Dr. Patricio Yankilevich, UTN-FRBA por introducirme en la Bioinformática. A mi directora la Dra. Gabriela Minetti, los jurados Dr. Gustavo Parisi, Dra. Silvana Fornasari y Dr. Galo Balatti, a los miembros de la UNQ en especial a la Dra. Carolina Cerrudo y a mis compañeros del cohorte 2017. Al Dr. Wim Vranken, todo el equipo de Bio2Byte, y al Dr. Rachid Tahzima de ILVO en Bélgica. A la familia de Restorando (luego TheFork), a Franco y Gabriel por todas las oportunidades junto a libertad de trabajar y estudiar la maestría a la vez. A Norberto por su apoyo mientras estudiaba ingeniería y trabajaba en ZL. A mis amigos que voy descubriendo en mi aventura de vivir en el extranjero, a Caroline y la familia Vankerkhoven. A mis grandes amigos que están siempre conmigo y a los familiares en Argentina que me siguen a la distancia.

Este trabajo final de maestría está dedicado a mi Abuela Inés.