ALGORITMOS DE PROGRAMACIÓN DINÁMICA CON R PARA RESOLVER PROBLEMAS DE ALINEAMIENTO DE SECUENCIAS

ÓSCAR SÁNCHEZ BECERRO

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería de Computadores

Madrid, 6 de febrero de 2014

Directora:
Victoria López López

Colaboradora de dirección:
Beatriz González Pérez

Autorización de Difusión

ÓSCAR SÁNCHEZ BECERRO

06/02/2014

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "ALGORITMOS DE PROGRAMACIÓN DINÁMICA CON R PARA RESOLVER PROBLEMAS DE ALINEAMIENTO DE SECUENCIAS", realizado durante el curso académico 2012-2014 bajo la dirección de Victoria López López del Departamento de Arquitectura de Computadores y Automática, y con la colaboración externa de Beatriz González Pérez del Departamento de Estadística e Investigación Operativa, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen

Este trabajo analiza e implementa mejoras consecuentes sobre algoritmos de alineamiento de secuencias basados en técnicas de Programación Dinámica. Como base fundamental se han utilizado algoritmos clásicos de alineamiento global Needleman-Wunsch y local Smith-Waterman respectivamente que han sido reprogramados por el autor en el lenguaje R para su optimización. Los algoritmos se han mejorado mediante técnicas que se explican en la memoria. Principalmente penalización por gaps y la utilización opcional de matrices de puntuación de aminoácidos que permiten valorar alineamientos. Este trabajo muestra alternativas de programación de los algoritmos con librerías dinámicas desarrolladas en lenguaje C y una comparativa de tiempos de ejecución. Además se ha desarrollado la plataforma multi-agente "MASBioseq" que permite distribuir y balancear la carga de los alineamientos de secuencias con mejoras de rendimiento que proporciona al usuario una interfaz gráfica accesible.

La razón fundamental de este desarrollo alternativo es comprobar la mejora del rendimiento que, como se demuestra en este trabajo, puede obtenerse con la compilación de determinadas partes del código en los distintos lenguajes. En esta memoria se desarrollan todas las temáticas necesarias para la comprensión del objetivo de los algoritmos y las técnicas utilizadas. Esto incluye un estado del arte sobre Bioinformática, Bioestadística, Programación Dinámica, alineamiento y análisis de secuencias biológicas, matrices de puntuación, técnicas y tipos de alineamiento, Agentes Inteligentes, entre otros.

Palabras clave

Programación dinámica, alineamiento de secuencias, algoritmos de alineamiento, librerías dinámicas, programación en R, agentes inteligentes, bioinformática, Needleman-Wunsch, Smith-Waterman.

Abstract

This document analyzes and implements consistent improvements on sequences alignment algorithms based on Dynamic Programming techniques. As a fundamental basis classic global alignment algorithm Needleman-Wunsch and local alignment algorithm Smith-Waterman have been used and reprogrammed for optimization respectively by the author in the R language. The algorithms have been improved using techniques explained in the document. Mainly gaps penalty and the optional use of amino acid scoring matrices that allow assessing alignments. This work presents an alternative for programming algorithms with dynamic libraries developed in C language and a comparison of execution times. It has also developed multi-agent "MASBioseq" platform that allows distribution and load balance of sequence alignments with performance enhancements that gives an accessible graphical user interface.

The reason of developing this alternative is to check the performance improvement, as demonstrated in this work, can be obtained with the compilation of certain parts of the code in different languages. In this document all necessary themes for understanding the objective of the algorithms and techniques used are developed. This includes a state of the art on Bioinformatics, Biostatistics, Dynamic Programming, alignment and analysis of biological sequences, scoring matrices, techniques and types of alignment, Intelligent Agents, among others.

Keywords

Dynamic programming, sequence alignment, alignment algorithms, dynamic libraries, R programming, intelligents agents, bioinformatics, Needleman-Wunsch, Smith-Waterman.

Índice de contenidos

Agradecimientos

La realización del presente trabajo final de máster, ha sido gracias a las sugerencias, orientaciones y estímulos de mis directoras Victoria López y Beatriz González, quienes me han estado guiando con profesionalidad a lo largo de todo el año académico con una actitud abierta y generosa, no sólo en el desarrollo del mismo, sino también en la adquisición de técnicas y metodologías investigadoras.

Quiero también dar las gracias al profesor Rubén Fuentes por orientarme en el uso de sistemas multi-agentes para mejorar los resultados de este trabajo, y a mis compañeros Jorge Martínez y Jorge Cordero por compartir conmigo experiencias y conocimientos.

Agradecer, por otra parte, a la Facultad de Informática y a la Universidad Complutense de Madrid por haberme dado la oportunidad de adquirir todos los conocimientos necesarios para la realización de esta labor de investigación.

Y, por supuesto, a mis familiares Mª Victoria Becerro, Aquilino Sánchez, Silvia Sánchez, Mª Dolores Becerro, Laureano Becerro y Josefa Hernández que supieron en todo momento respetar y sobrellevar mi ausencia durante todo el periodo académico.

1. Introducción

El objetivo de este trabajo fin de máster es desarrollar una serie de funciones para resolver problemas de alineamiento de secuencias utilizando técnicas de Programación Dinámica. Estas funciones, se encuentran actualmente empaquetadas en la librería 'Bioseq' [1] tomando como base fundamental los algoritmos clásicos de Needleman-Wunsch y Smith-Waterman que han sido reprogramados por el autor de este trabajo empleando el entorno y el lenguaje R. Los algoritmos se han mejorado mediante la inclusión de características de sobrecarga en los parámetros que permiten no sólo la penalización por gaps sino también la utilización opcional de matrices de puntuación (PAM y BLOSOM) de aminoácidos permitiendo optimizar la valoración de los alineamientos. Además, se han desarrollado alternativas de programación de los algoritmos mostrados mediante el uso de librerías dinámicas desarrolladas en lenguaje C y compiladas con MinGW. La razón fundamental de este desarrollo alternativo es comprobar la mejora del rendimiento que, como se demuestra en este trabajo, puede obtenerse con la compilación de determinadas partes del código en los distintos lenguajes.

Para el desarrollo satisfactorio de este trabajo se han utilizado los conocimientos adquiridos principalmente en las asignaturas de Bioinformática y de Agentes Inteligentes impartidas en el Máster de Investigación en Informática donde se enmarca este trabajo.

La Bioinformática proviene de la informática y de la biología, que involucra la tecnología computacional para el almacenamiento, recuperación, manipulación y distribución de la información relacionada con las macromoléculas biológicas como el ADN, ARN y proteínas. Uno de los mayores retos de esta disciplina, se encuentra en la limitada capacidad y sofisticación de la mayoría de los algoritmos para reflejar realmente la realidad; lo que conlleva a predicciones incorrectas que no tienen sentido en contextos biológicos. Además, las limitaciones computacionales obligan a los investigadores utilizar herramientas bioestadísticas que posibilitan el desarrollo de algoritmos menos exhaustivos pero más veloces [2].

Este campo, no obstante, cuenta con el esfuerzo que están desempeñando diversos laboratorios e instituciones, no sólo con el fin de recoger datos biológicos y almacenarlos en grandes bases de datos como pueden ser GenBank [3] y UniProt [4], sino también con el objetivo de investigar y mejorar las diferentes técnicas y herramientas dentro de las limitaciones actuales. Gracias a todos estos avances, los investigadores podrán obtener resultados utilizando y sacando conclusiones de los datos mediante el análisis y el diseño de algoritmos cada vez más sofisticados aplicables a muchos otros campos (economía, industria, etc).

Las principales aportaciones de este trabajo son las siguientes:

1. Programación de los algoritmos de alineamiento en R y su mejora con las librerías dinámicas (DLLs).
2. Desarrollo del sistema multi-agente "MASBioseq" para la implementación de una interfaz gráfica que permita la paralelización de los procesos involucrados y de los equipos físicos necesarios necesarios para obtener resultados eficientes.
3. Análisis comparativo de los resultados en diferentes entornos.
4. Desarrollo de un CD con el siguiente contenido:
    - Algoritmos de alineamiento de secuencias tanto globales como locales programados en R junto con las versiones que ejecutan librerías dinámicas programadas en lenguaje C.
    - Ejecutable de la plataforma multi-agente "MASBioseq" para Windows.
    - Clases JAVA utilizadas para la implementación del sistema multi-agente.
    - Algoritmos de alineamiento de secuencias iniciales modificados con el fin de adaptarlos a la plataforma "MASBioseq".
    - Material utilizado durante la defensa y exposición del proyecto.

Este trabajo se estructura de la siguiente forma: El capítulo 2 introduce los fundamentos de la Bioinformática y la Bioestadística. El capítulo 3, presenta el lenguaje R sobre el cual se han realizado los algoritmos y pruebas, además del proyecto Bioconductor, muy relacionado con el análisis de secuencias que nos compete en este trabajo. El análisis de secuencias se introducen en

el capítulo 4. En el capítulo 5 se explican los métodos de análisis de secuencias. En el capítulo 6 se presenta una plataforma distribuida de agentes inteligentes que permite el balanceo y la distribución de carga en los alineamientos de secuencias. En el capítulo 7 se muestran los resultados obtenidos al comparar los desarrollos basados en R y los basados en C con DLLs, además de los códigos aportados por el autor para generalizar los alineamientos mediante las matrices de puntuación. Finalmente, las conclusiones y el trabajo futuro se muestran en el capítulo 8.

2. Bioinformática y Bioestadística

En este capítulo se muestran los conceptos más importantes sobre Bioinformática y Bioestadística por ser fundamentales para la comprensión y el desarrollo del trabajo realizado en esta memoria. También se introduce la relación estrecha existente entre estas dos disciplinas.

2.1. Bioinformática

La bioinformática, es la disciplina de análisis cuantitativo de la información relativa a las macromoléculas biológicas con la ayuda de computadoras. El desarrollo de la bioinformática como campo, es el resultado de los avances en biología molecular e informática en los últimos 30-40 años. A pesar de que estos avances no se describen en detalle aquí, la comprensión de la historia de esta disciplina es útil para obtener una visión más amplia en la investigación bioinformática actual. Un resumen cronológico breve de los acontecimientos históricos que han tenido un gran impacto en el desarrollo de la bioinformática se presenta aquí para proporcionar un contexto.

Los primeros intentos de la bioinformática se remontan a la década de 1960, aunque la palabra bioinformática no existiera entonces. Probablemente, el primer gran proyecto de bioinformática fue realizada por Margaret Dayhoff en 1965, que desarrolló una primera base de datos de secuencias de proteínas llamada "Atlas de secuencias y estructuras de proteínas". Posteriormente, a principios de 1970, el Laboratorio Nacional de Brookhaven creó el Banco de Datos de Proteínas para el almacenamiento de estructuras de proteínas tridimensionales. En sus inicios, la base de datos almacenaba menos de una docena de estructuras de proteínas, en comparación con las más de 30.000 estructuras actuales. El primer algoritmo de alineamiento de secuencias fue desarrollado por Needleman y Wunsch en 1970. Este fue un paso fundamental en el desarrollo del campo de la bioinformática, que allanó el camino en la comparación de secuencias rutinarias y en la búsqueda de bases de datos practicadas por los biólogos modernos. El primer algoritmo de predicción de estructuras de proteínas fue desarrollado por Chou y Fasman en 1974. A pesar de ser bastante rudimentario (según los estándares de hoy en día) fue

pionero en una serie de avances en la predicción de estructuras proteicas. La década de 1980 vio el establecimiento de GenBank y el desarrollo de algoritmos de búsqueda rápida de bases de datos, tales como FASTA de William Pearson y BLAST por Stephen Altschul et al. El inicio del proyecto del genoma humano a finales de 1980 proporcionó un gran impulso para el desarrollo de la bioinformática. El desarrollo y el uso cada vez más generalizado de Internet en la década de 1990 hizo posible el intercambio, la difusión y el acceso instantáneo a los datos biológicos.

Estos son sólo los principales hitos en la creación de este nuevo campo. La razón fundamental por la que la bioinformática ganó importancia como disciplina, fue por el avance de los estudios del genoma que produjeron cantidades sin precedentes de datos biológicos. La explosión de información de secuencias genómicas ha generado una demanda repentina de herramientas computacionales eficientes para gestionar y analizar los datos. El desarrollo de estas herramientas computacionales dependía de los conocimientos generados a partir de una amplia gama de disciplinas como matemáticas, estadística, informática, tecnologías de la información y biología molecular. La fusión de estas disciplinas ha creado un campo orientado a la información en biología, que ahora se conoce como bioinformática.

La bioinformática es un área de investigación interdisciplinaria que sirve de conexión entre las ciencias computacionales y las ciencias biológicas. Existe una variedad de definiciones tanto en la literatura como en la red informática mundial; siendo algunas más inclusivas que otras. En este caso, adoptaremos la definición propuesta por Luscombe et al. que define la bioinformática como la unión entre la informática y la biología: la bioinformática involucra la tecnología que utilizan las computadoras para el almacenamiento, recuperación, manipulación y distribución de la información relacionada con las macromoléculas biológicas como el ADN, ARN y proteínas. El énfasis se encuentra en el uso de computadoras, ya que la mayoría de las tareas orientadas al análisis de datos genómicos son muy repetitivas o matemáticamente complejas. El uso de las computadoras es absolutamente indispensable en la minería de los genomas para la recolección de información y en la construcción de conocimiento.

La bioinformática difiere de un campo relacionado conocido como biología computacional, ya que la bioinformática se limita a la secuencia, al análisis funcional y estructural de los genes, genomas y sus productos correspondientes y, a menudo, se considera también a la biología computacional a nivel molecular. Sin embargo, la biología computacional abarca todas las áreas biológicas que implican cálculo. Por ejemplo, el modelado matemático de los ecosistemas, la dinámica de la población, la aplicación de la teoría de juegos en los estudios de comportamiento y la construcción filogenética utilizando los registros fósiles. Todas emplean herramientas computacionales, lo que no implica necesariamente macromoléculas biológicas.

Junto a esta distinción, cabe señalar que hay otros puntos de vista de cómo los dos términos se relacionan. Por ejemplo, una versión define la bioinformatica como el desarrollo y aplicación de herramientas computacionales en la gestión de todo tipo de datos biológicos, mientras que la biología computacional está más limitada a la elaboración teórica de los algoritmos utilizados para la bioinformática. La confusión en la actualidad sobre la definición puede reflejar en parte la naturaleza de este nuevo campo dinámico y en rápida evolución.

El objetivo final de la bioinformática es comprender mejor una célula viva y cómo funciona a nivel molecular. Mediante el análisis de secuencia molecular y datos estructurales, la investigación bioinformática puede generar nuevas ideas y proporcionar una perspectiva "global" de la célula. La razón por la que las funciones de una célula se puede entender mejor mediante el análisis de la secuencia de datos es en última instancia debido a que el flujo de la información genética está dictada por el "dogma central" de la biología en la cual el ADN se transcribe en ARN, que se traduce a proteínas. Las funciones celulares se llevan a cabo principalmente por proteínas cuyas capacidades se determinan en última instancia por sus secuencias. Por lo tanto, la solución de problemas funcionales utilizando la secuencia, y a veces enfoques estructurales, ha demostrado ser una tarea fructífera.

La bioinformática consta de dos subcampos:

- El desarrollo de herramientas y bases de datos.
- La aplicación de estas herramientas y bases de datos en la generación de conocimientos biológicos para comprender mejor los sistemas vivientes.

Estos dos subcampos se complementan entre sí. El desarrollo de herramientas incluye la escritura de software para la secuencia, análisis estructural y funcional, así como la construcción y comisariado de bases de datos biológicos. Estas herramientas se utilizan en tres áreas de la investigación genómica y de la biología molecular:

- Análisis de la secuencia molecular.
- Análisis estructural molecular.
- Análisis funcional molecular.

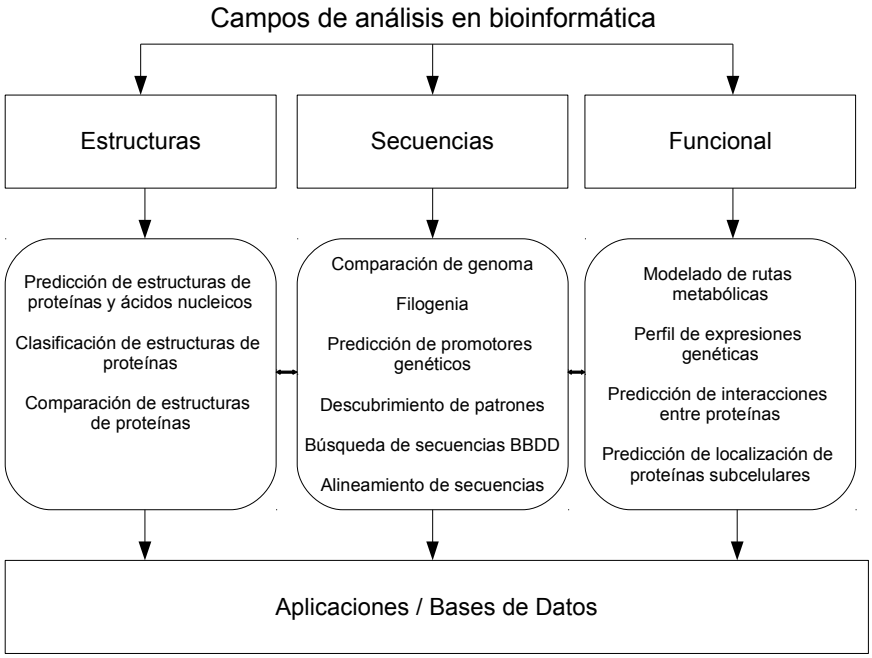


Figura 1. Resumen de varios subcampos de la bioinformática.

Los análisis de los datos biológicos a menudo generan nuevos problemas y desafíos que a su vez estimulan el desarrollo de nuevas y mejores herramientas computacionales. Las áreas de análisis de secuencias incluyen el alineamiento de secuencias, búsqueda de secuencias en bases de datos, motivos y patrones de descubrimiento, descubrimiento de genes y promotores, reconstrucción de las relaciones evolutivas, y el ensamblaje del genoma y su comparación. Los análisis estructurales incluyen el análisis de proteínas, análisis de estructuras de ácidos nucleicos, comparación, clasificación y predicción. Los análisis funcionales incluyen pruebas de expresión génica, interacción y predicción entre proteínas, predicción de la localización subcelular de las proteínas, reconstrucción de la secuencia metabólica, y la simulación.

Los tres los aspectos de análisis de la bioinformática no están aislados ya que a menudo interactúan para producir resultados integrados. Por ejemplo, la predicción de la estructura proteica depende del alineamiento de secuencias de datos; la agrupación de los perfiles de expresión génica requiere la utilización de métodos de construcción de árboles filogenéticos obtenidos en el análisis de secuencia. La secuencia basada en predicciones promotoras está relacionada con el análisis funcional de los genes coexpresados. La anotación génica consiste en una serie de actividades, que incluyen distinción entre codificación y secuencias no codificadas, la identificación de secuencias de proteínas traducidas, y la determinación de la relación evolutiva del gen con otros genes conocidos; la predicción de sus funciones celulares emplea herramientas de los tres grupos de análisis.

La bioinformática no sólo se ha convertido en algo esencial para la investigación en genómica básica y biología molecular, sino que también está teniendo un gran impacto en muchas áreas de la biotecnología y de las ciencias biomédicas. Tiene aplicaciones, por ejemplo, en el diseño basado en el conocimiento de drogas, el análisis forense de ADN, y la biotecnología agrícola. Estudios computacionales de las interacciones entre proteínas ligadas proporcionan una base racional para la rápida identificación de nuevas derivaciones potenciales para las drogas sintéticas. El conocimiento de las estructuras tridimensionales de las proteínas permite que las moléculas sean diseñadas de forma que sean capaces de vincularse en la ubicación del receptor de una proteína diana con gran afinidad y especificidad.

Este enfoque basado en la informática reduce significativamente el tiempo y el coste necesario para el desarrollo de fármacos con mayor potencia, menos efectos secundarios, y menos toxicidad que usando el enfoque de ensayo y error tradicional. En medicina forense, los resultados del análisis filogenético molecular se han aceptado como prueba en los tribunales penales. Algunas estadísticas Bayesianas sofisticadas y métodos basados en la verosimilitud para el análisis de ADN se han aplicado en el análisis de la identidad forense. Es importante mencionar que la genómica y la bioinformática están a punto de revolucionar nuestro sistema de salud mediante el desarrollo de la medicina personalizada. La secuenciación del genoma de alta velocidad junto con la sofisticada tecnología informática permitirá a un médico en una clínica secuenciar rápidamente el genoma de un paciente y detectar fácilmente posibles mutaciones dañinas y participar en el diagnóstico precoz y en el tratamiento eficaz de las enfermedades. Las herramientas bioinformáticas están siendo utilizadas también en la agricultura. Las bases de datos genómicas de la flora y los análisis de expresión del perfil génico, han jugado un papel importante en el desarrollo de nuevas variedades de cultivos que tienen una mayor productividad y una mayor resistencia a las enfermedades.

Tras reconocer el poder de la bioinformática, también es importante tener en cuenta sus limitaciones y evitar la excesiva dependencia y el exceso de confianza de la producción bioinformática. De hecho, tiene una serie de limitaciones inherentes.

La bioinformática y la biología experimental son independientes, pero a su vez, con actividades complementarias. La bioinformática depende de la ciencia experimental para producir datos primarios para el análisis. Esto, a su vez, proporciona interpretación útil de los datos experimentales y lo más importante es, que conduce hacia la investigación experimental. Las predicciones en bioinformática no son pruebas formales de los conceptos. No sustituyen a los métodos tradicionales en la investigación experimental sobre los actuales test de hipótesis. Además, la calidad de las predicciones en bioinformática depende de la calidad de los datos y de la sofisticación de los algoritmos que se han utilizado. Los datos secuenciados a través del análisis con alto volumen de información a menudo contienen errores. Si las secuencias son erróneas o contienen anotaciones incorrectas, los resultados de los análisis resultantes son

engañosos también. Por eso, es tan importante mantener una perspectiva realista del papel de la bioinformática.

La bioinformática no es de ninguna manera un campo maduro. La mayoría de los algoritmos no tienen la capacidad ni la sofisticación necesaria para reflejar realmente la realidad. A menudo hacen predicciones incorrectas que no tienen sentido cuando se coloca en un contexto biológico. Los errores en la secuencia de alineamiento, por ejemplo, pueden afectar al resultado de un análisis estructural o filogenético. El resultado computacional también depende de la potencia disponible. Muchos algoritmos precisos pero exhaustivos, no se pueden utilizar debido a la velocidad lenta de computación. En su lugar, los algoritmos más rápido pero menos precisos tienen que ser utilizados. Esta es una solución de compromiso entre la precisión necesaria y la viabilidad computacional. Por lo tanto, es importante tener en cuenta el potencial de errores producidos por los programas bioinformáticos. La precaución debe de ser siempre ejercida al interpretar los resultados de la predicción. Es una buena práctica utilizar varios programas, si están disponibles, y realizar múltiples evaluaciones. Una predicción más precisa a menudo se puede obtener si uno dibuja un consenso mediante la comparación de los resultados de diferentes algoritmos.

A pesar de las dificultades, no hay duda de que la bioinformática es un campo que tiene un gran potencial para revolucionar la investigación biológica en las próximas décadas. En la actualidad, el campo está experimentando una importante expansión. Además de proporcionar herramientas computacionales más confiables y más rigurosas para el análisis secuencial, estructural y funcional, el principal desafío para el futuro desarrollo de la bioinformática es el desarrollo de herramientas para esclarecer las funciones y las interacciones de todos los productos genéticos de una célula. Esto presenta un enorme desafío debido a que requiere la integración de los diferentes campos de conocimiento biológico y una variedad de herramientas matemáticas y estadísticas complejas. Para obtener una comprensión más profunda de las funciones celulares, se necesitan modelos matemáticos para simular una amplia variedad de reacciones intracelulares e interacciones a nivel celular.

A esta simulación molecular de todos los procesos celulares se le denomina biología de sistemas. El logro de este objetivo supondría un salto importante hacia la plena comprensión de un sistema viviente. Por eso, la simulación y la integración a nivel de sistema son considerados como el futuro de la bioinformática. El modelado de redes complejas y realizar predicciones sobre su comportamiento presentan enormes desafíos y oportunidades para los bioinformáticos. El objetivo final de esta iniciativa es la de transformar la biología desde una ciencia cualitativa a una ciencia cuantitativa y predictiva [5].

2.2. Bioestadística

La bioestadística es una rama de la estadística que se ocupa de los problemas planteados dentro de las ciencias de la vida, abarcando el diseño de experimentos biológicos (especialmente en la medicina, en la agricultura y en la pesca), la recolección, el resumen y el análisis de los datos de los experimentos, la interpretación y la inferencia de los resultados. En bioinformática, por ejemplo, es utilizada en secuencias genómicas con el objetivo de encontrar estructuras de interés (e.j. genes) en determinados tipos de secuencias:

- Muy largas (millones de elementos en cada secuencia).
- Sin información relevante desde el punto de vista biológico.
- Difíciles de distinguir del ruido.

Para lograrlo, existe la necesidad de utilizar herramientas sofisticadas de estadística (modelos probabilísticos) y de minería de datos (algorítmica e inteligencia artificial). Un modelo, es en el fondo una propuesta que trata de encontrar un patrón en la forma en la que se distribuyen las secuencias reales, que nos servirá para estudiar la estructura interna, la similitud y la evolución de las secuencias. Primeramente se propone un modelo para una secuencia real dada y se ve si el modelo se ajusta a la realidad. Si se ajusta, nos da una explicación parcial de su comportamiento; si no, bien se descarta, o se reajustan los parámetros para ver si de esta manera termina ajustándose. En la Tabla 1 se muestra un resumen de los modelos probabilísticos utilizados para la detección de homologías.

Tabla 1. Resumen de los modelos probabilísticos.

Modelo	Asunción	Elementos Independientes	Probabilidad de iguales
Multinomial	Frecuencias iguales	Sí	Sí
Multinomial	Frecuencias distintas, basadas en alguna evidencia biológica.	Sí	No
Markov	Cada nucleótido depende de los *n* anteriores (orden *n*)	No	No

Un modelo siempre nos dará un resultado al que tendremos que dar un significado en base a si la cadena corresponde o no con el modelo aplicado. Estadísticamente se determina la probabilidad de que sí se corresponda o que la correspondencia observada sea solamente fruto del azar. Biológicamente, se buscan razones biológicas para la adecuación (o no) del modelo a la secuencia. En principio, se busca determinar si el patrón (del tipo que sea) que hemos encontrado es o no fruto del azar. Hay distintas opciones, pero básicamente se trata de comparar la frecuencia observada con la frecuencia esperada según un determinado modelo. En definitiva, nos define la probabilidad de que lo encontrado se parezca o no al modelo. Si el modelo es una distribución aleatoria, determinaremos si la frecuencia observada es fruto o no del azar. Si el modelo es una distribución definida, en cambio, determinaremos si la frecuencia observada sigue o no dicho modelo. Al encontrar un patrón en una secuencia, debemos considerar que lo podríamos haber encontrado por casualidad, al igual que sucede en una secuencia aleatoria:

- P-valor p: probabilidad (entre 0 y 1) de que lo hayamos encontrado por casualidad. Típicamente, se admiten entre un 5% y un 0.1% de fallos. El umbral de admisión se suele llamar α.

- $p<\alpha$: se rechaza la hipótesis nula → no es fruto del azar.
- $p>\alpha$: se acepta la hipótesis nula → es fruto del azar.
- α suele ser 0.05, 0.01, 0.001 (5%, 1% ó 0.1%).

- Hipótesis nula (H_0): Un patrón ha sido encontrado al azar.
- Hipótesis alternativa: no ha sido encontrado al azar.

Para saber lo probable que es encontrar un patrón al azar, tenemos que diseñar una técnica para generar secuencias (o en general, muestras) aleatorias. Podríamos suponer una distribución aleatoria de nucleótidos (bien suponiendo las mismas probabilidades para cada nucleótido o para las frecuencias conocidas para un determinado organismo) o podríamos reordenar aleatoriamente los nucleótidos (elementos) de la secuencia (muestra) real, que asegura mantener las propiedades estadísticas de la secuencia real. Una vez elegido el método de aleatorización, generamos *N* muestras aleatorias y calculamos la similitud de nuestro patrón con respecto a las secuencias. Idealmente, las puntuaciones aleatorias seguirán una distribución normal o gaussiana, donde tendremos una media de las puntuaciones aleatorias (μ) y una desviación estandar de dichas puntuaciones (σ), aunque no siempre sea así. Podemos calcular lo desviada que está la puntuación sobre la secuencia real (x) respecto a las puntuaciones sobre secuencias aleatorias, mediante el *Z*-score:

$$Z=\frac{x-\mu}{\sigma}$$

Mediante tablas estadísticas, determinamos la probabilidad de que en una distribución aleatoria tengamos un valor como $x \rightarrow$ *p-valor.* Hay que tener en cuenta, que existe la necesidad de compromiso entre sensibilidad y especificidad, y que es mejor dar un falso negativo que dar un falso positivo:

- Falso positivo: aceptar un patrón que se debe al azar.
- Falso negativo: rechazar un patrón que no se debe al azar.
- Sensibilidad: capacidad de detectar todos los patrones que no se deben al azar.
- Especificidad: capacidad de descartar todos los patrones que se deben al azar.

En relación a los múltiples contrastes de hipótesis, si comparamos un patrón con una secuencia, y tiene un *p-valor* de 0.001, quiere decir que hay una probabilidad de un 0.1% de que se deba al azar (por lo tanto, el patrón es bueno). Si comparamos el patrón con un millón de secuencias, tenemos un millón de oportunidades de obtener un *p-valor* bajo (con la consiguiente necesidad de corregir los umbrales de significación α). En la Tabla 2 se muestra la corrección de los p-valores para diferentes métodos estadísticos.

Tabla 2. Corrección de los p-valores en diferentes métodos estadísticos.

Corrección	Significado (α=0.01)	Especificidad
Bonferroni	Como mucho un 1% serán falsos positivos	Muy alta
FWER	Probabilidad de un 1% de tener al menos un falso positivo.	Alta
FDR	Exactamente un 1% serán falsos positivos.	Moderada

El *odds ratio* es una manera de calcular la probabilidad similar a la terminología de las apuestas (20 a 1). Es una medida de lo probable que es un suceso coincidente (GC) teniendo en cuenta lo probable que son los sucesos por separado (G,C).

$$odds\ ratio \simeq \frac{N(x,y)}{N(x)N(y)}$$

- N(xy) = Frecuencia observada.
- N(x) N(y) = Frecuencia esperada (dependerá del modelo de referencia, en este caso para el multinomial).

Por último, mencionar el modelo oculto de Markov (HMM), que es un modelo de Markov en el que no podemos observar los estados directamente aunque sean conocidos; pero podemos inferirlos a partir de observaciones. La secuencia se modela como si fuera generada por una cadena de Markov. En cada posición tenemos uno o más estados desconocidos (ocultos), observando únicamente los símbolos de la secuencia generados de acuerdo a una distribución multinomial que depende de dichos estados desconocidos. El objetivo es que a partir de la secuencia observada (ruidosa) se infieran los estados ocultos.

En definitiva, el uso de modelos distorsiona (simplifica) la realidad, pero nos ayuda a entenderla en parte. Es importante un equilibrio en la complejidad del modelo, ya que un modelo demasiado simple puede no ser útil, pero uno muy complicado puede estar muy influenciado por datos externos. Un modelo es siempre una guía, y la adecuación o no a un patrón debe siempre estimarse según su significado estadístico, y corroborarse según evidencias biológicas. Nunca es una prueba irrefutable de algo. La significación estadística debe ser rigurosa para minimizar el número de falsos positivos y negativos. Ante la duda, suele ser recomendable ser conservadores en nuestras afirmaciones. Es importante tener en cuenta el número de pruebas (si hay más de una) para realizar correcciones a los estadísticos. Los modelos ocultos son un tipo de modelos bastante utilizados en bioinformática para determinar el patrón cuando no se pueden hacer asunciones del modelo a priori, siendo muy utilizados en alineamientos [6].

3. Proyecto R y Bioconductor

En este capítulo se presenta una introducción al proyecto R y Bioconductor con el fin de ayudar al lector a comprender los conceptos, las funciones y los paquetes que ha utilizado el autor para la reimplementación de los algoritmos clásicos, incluyendo todas las características que considera de utilidad para futuros proyectos.

Comenzamos con R [7], que es un lenguaje de programación libre, un proyecto y un entorno de desarrollo para el análisis estadístico y gráfico. Como lenguaje, R es un dialecto del lenguaje S, un lenguaje estadístico orientado a objetos desarrollado a finales de la década de 1980 por AT&T's Bell Labs. El proyecto R ha sido el resultado de un largo esfuerzo académico donde la mayoría de sus contribuidores han sido estadísticos. Este proyecto comenzó en 1995 por un grupo de estadísticos de la Universidad de Auckland y que, desde entonces, ha seguido creciendo. Debido a que la estadística es una ciencia interdisciplinaria, el uso de R ha atraído a numerosos investigadores académicos que ejercen en varios campos de la estadística aplicada como pueden ser en la estadística ambiental, econometría, medicina y aplicaciones de salud pública, y bioinformática entre otras [8].

Bioconductor [9] es un proyecto de código abierto para el análisis y comprensión de los datos genómicos generado en laboratorios de experimentación en biología molecular. Se basa principalmente en el lenguaje de programación estadístico R, pero contiene contribuciones en otros lenguajes de programación. Cada año lanza dos versiones que sigue al lanzamiento de las versiones semestrales de R. En un momento dado, hay una versión de lanzamiento de R, y una versión de desarrollo. La mayoría de los usuarios encuentran la versión de lanzamiento adecuada para sus necesidades habiendo, además, un gran número de paquetes de anotación del genoma disponibles que están principalmente, pero no exclusivamente, orientados hacia diferentes tipos de microarrays. El proyecto se inició en otoño de 2001 y está supervisado por el equipo principal de Bioconductor, del Centro de Investigación del Cáncer “Fred Hutchinson”, junto con otros miembros provenientes de diversas instituciones de Estados Unidos e internacionales [10].

Actualmente, CRAN (The Comprehensive R Archive Network) dispone también de un repositorio al igual que Bioconductor, con una gran variedad de paquetes. En nuestro caso**,** instalaremos y trabajaremos con el paquete "SequinR" (recuperación y análisis de secuencias biológicas) [11], que es comúnmente utilizado para el análisis exploratorio de datos y visualización de secuencias de datos biológicas (ADN y proteínas).

3.1. Proyecto R

R es un lenguaje de programación estadístico de código abierto que es utilizado para manipular datos, para realizar análisis estadísticos, y para presentar resultados gráficos entre otras tareas. R consiste en un lenguaje principal que dispone de una serie de paquetes que son distribuidos con el lenguaje, mientras que otros son aportados por la comunidad. Los paquetes agregan una funcionalidad específica a la instalación de R, convirtiéndose gracias a ello, en el lenguaje principal de análisis estadístico académico; ampliamente utilizado en diversas áreas de investigación, del gobierno, y la industria.

R tiene varias características únicas. Tiene una interfaz antigua (aunque si se desea, se puede sustituir por Tinn-R [12] o por Revolution R [13] que contiene una interfaz más amigable) donde los usuarios escriben comandos en una consola. Los scripts en texto plano representan flujos de trabajo disponiendo R de muchas herramientas distintas para la edición (y otras tareas). Al ser un lenguaje de programación flexible, permite que mientras una persona utiliza las funciones proporcionadas por R para conseguir tareas analíticas avanzadas, otra pueda implementar sus propias funciones para nuevos tipos de datos. Como lenguaje de programación, R adopta una sintaxis y una gramática que difiere de muchos otros lenguajes: los objetos en R son vectores, y las funciones son 'vectorizadas' para operar con todos los elementos del objeto. Los objetos en R tienen una semántica de copia sobre el cambio y de paso por valor, reduciendo las consecuencias inesperadas para los usuarios en detrimento del uso menos eficiente de la memoria. Es importante puntualizar que los paradigmas comunes en otros lenguajes, como el bucle '`for`', no son tan comunes en R.

Al abrir una sesión en R, nos encontramos con el "prompt" que es donde el usuario escribirá y ejecutará las diferentes instrucciones. A continuación se describen las principales características que han sido necesarias para este trabajo.

Tipos de datos esenciales en R: R tiene un número de tipos de datos estándar, para representar datos `integer`, `numeric` (de coma flotante), `complex`, `character`, `logical` (Boolean), `raw` (byte). Es posible la conversión entre tipos de datos, y el descubrimiento del tipo de dato de una variable. En la Tabla 3 se muestran los tipos de datos esenciales en R.

Tabla 3. Tipos de datos esenciales en R.

Entrada	Salida	Tipo
`c(1.1, 1.2, 1.3)`	`1.1 1.2 1.3`	`numeric`
`c(FALSE, TRUE, FALSE)`	`FALSE TRUE FALSE`	`logical`
`c("foo", "bar", "baz")`	`"foo" "bar" "baz"`	`character`

R incluye tipos de datos muy útiles para el análisis estadístico, incluyendo `factor` para representar a las categorías y `NA` (utilizado en cualquier vector) para representar los valores perdidos.

Listas, tramas de datos, y matrices: Todos los vectores mencionados hasta ahora son homogéneos, es decir, que consta de un solo tipo de elemento. Una lista puede contener un conjunto de diferentes tipos de elementos y, al igual que todos los vectores, estos elementos pueden ser nombrados para crear una asociación clave-valor.

Un `data.frame` es una lista de vectores de igual longitud, que representa una estructura de datos rectangular. Cada columna de la trama de datos es un vector, por lo que los tipos de datos deben ser homogéneos dentro de una columna. Un `data.frame` puede ser un subconjunto de fila o columna, y las columnas pueden ser accedidas con `$` o `[[`.

Una `matrix` es una estructura de datos para la representación de datos homogéneos rectangulares en dimensiones más altas. En la Tabla 4 se muestran ejemplos de listas, tramas de datos, y matrices.

Tabla 4. Listas, tramas de datos, y matrices.

Listas	Tramas de datos	Matrices
`List (a=1:3, b=c("foo", "bar"))`	`df <- data.frame (age=c(27L, 32L, 19L), sex=factor (c("Male","Female", "Male")))`	`m<-matrix(1:12,nrow=3)`
`$a 1 2 3 $b "foo" "bar"`	`age sex 1 27 Male 2 32 Female 3 19 Male`	`[,1] [,2] [,3] [,4] [1,] 1 4 7 10 [2,] 2 5 8 11 [3,] 3 6 9 12`

Clases S3 y S4: Son estructuras de datos más complejas que se representan mediante el sistema de objetos 'S3' o 'S4'. Muchos paquetes de Bioconductor implementan objetos S4 para representar datos. Los sistemas 'S3' y 'S4' son bastante diferentes desde la perspectiva de un programador, pero bastante similar desde la perspectiva de un usuario: ambos sistemas encapsulan estructuras de datos complejas y permiten métodos especializados para diferenciar los tipos de datos diferentes; se utilizan funciones de acceso para extraer la información de los objetos.

Funciones: Las funciones en R aceptan argumentos y retornan valores. Los argumentos pueden ser obligatorios u opcionales. Algunas funciones pueden tener un número variable de argumentos.

R tiene un muy gran número de funciones. La Tabla 5 muestra una breve lista de las funciones más comunes que han sido de utilidad para este trabajo.

Tabla 5. Diferentes funciones en R junto con su descripción.

Funciones	Descripción
`dir, read.table (y similares), scan`	Lista archivos, lee hojas de cálculo, lee eficientemente datos homogéneos para representarlos como matriz.
`c, factor, data.frame, matrix`	Crea un vector, marco de datos o matriz.
`summary, table, xtabs`	Resume, crea una tabla el número de veces de los elementos del vector, tabulación cruzada entre variables.
`t.test, aov, lm, anova, chisq.test`	Comparación básica de grupos.
`dist, hclust`	Datos clúster.
`plot`	Datos gráficos.
`ls, str, library, search`	Lista objetos en el área de trabajo actual (o específica), o alcanza la cumbre de una estructura de un objeto; añade una librería o describe la ruta de búsqueda de paquetes adjuntos.
`lapply, sapply, mapply, aggregate`	Aplica una función a cada elemento de una lista (`lapply`,`sapply`), a varias listas (`mapply`), o a elementos de una lista particionada por factores (`aggregate`).
`with`	Acceso a columnas sin tener que repetir el nombre de la trama de datos.
`match, %in%`	Reporta el índice o la existencia de elementos de un vector que responden a otro.
`split, cut`	Divide un vector por un factor de igual longitud, corta un solo vector en intervalos codificados como niveles de un factor.

`strsplit, grep, sub`	Opera sobre vectores de caracteres, dividiéndolo en distintos campos en busca de patrones usando expresiones regulares.
`install.packages`	Instala un paquete desde el repositorio.
`traceback, debug, browser`	Reporta la secuencia de funciones bajo evaluación en el momento del error.

Los paquetes proporcionan funcionalidades más allá de la disponible de base en R. Hay más de 4000 paquetes en CRAN (Comprehensive R Archive Network) y más de 600 paquetes de Bioconductor. Los paquetes son aportados por diversos miembros de la comunidad; variando en calidad (muchos son excelentes) y conteniendo a veces aspectos idiosincrásicos en su implementación.

La realización de paquetes es muy costosa, porque a pesar de existir muchos documentos de referencia para la construcción de paquetes, en nuestro grupo hemos constatado que ninguno de ellos es realmente útil por su incompletitud, que conduce inevitablemente a una gran cantidad de errores y warnings. Sin embargo, gracias al trabajo desarrollado por Jorge Martínez [1], disponemos de un manual completamente elaborado para realizar desarrollos seguros siguiendo las normativas de aceptación del proyecto CRAN. A continuación, se muestra en la Tabla 6 un resumen de los paquetes básicos fundamentales junto con una selección de paquetes contribuidos.

Tabla 6. Paquetes básicos fundamentales y contribuidos junto con sus descripciones.

Paquete	Descripción
base	Datos de entrada y manipulación esencial; conceptos de programación y scripting.
stats	Funciones estadísticas y gráficas esenciales.
lattice, ggplot2	Aproximación a gráficos avanzados.
methods	Métodos y clases para 'S4'.
parallel	Facilidades para evaluación paralela.

Para encontrar ayuda mediante el sistema de ayuda de R, hay que iniciar el navegador web desde el 'prompt' con:

```
> Help.start()
```

Manuales: Se pueden usar las páginas del manual para encontrar descripción detallada de los argumentos y de los valores devueltos por las funciones, estructuras y métodos de clases. Para encontrar información dentro de una sesión de R, hay que escribir `'?'` antes de la función.

```
> ?data.frame
```

Viñetas: Las viñetas, especialmente en los paquetes de Bioconductor, proporcionan una amplia narrativa que describen la funcionalidad global del paquete.

```
> vignette(package="EMBO2012")
```

Hay a menudo muchas maneras de lograr un resultado en R, pero esas maneras distintas a menudo difieren en velocidad o en requisitos de memoria. Para pequeños conjuntos de datos, el rendimiento no es tan importante, pero para grandes conjuntos de datos (ej. la secuenciación de alto rendimiento, estudios de asociación de grandes genomas, GWAS) o cálculos complicados (ej. bootstrapping) el rendimiento puede ser importante. Es por eso que existe la posibilidad de implementar en pocas líneas de R programas que en C involucrarían algunas cientos de ellas. Los programas escritos en R reflejan de una manera más clara los algoritmos estadísticos que implementan, a diferencia de C que tiende a desdibujarlos. Además, cuenta con la ventaja de que es más fácil de depurar código en R por ser un lenguaje interpretado. Sin embargo, R adolece de los problemas típicos de los lenguajes interpretados; es bastante ineficiente para resolver problemas que exigen un alto coste computacional.

La solución a este problema se encuentra, en muchos casos, en incorporar a los programas escritos en R rutinas escritas en otros lenguajes de programación compilados. Una de las posibilidades que contempla R, es la de utilizar funciones contenidas en una DLL escrita en C.

Una DLL es, en traducción literal del acrónimo, una biblioteca (o librería) de enlace dinámico. Una DLL contiene cierto número de funciones compiladas que no se ejecutan autónomamente, sino que son invocadas por otros programas. Dentro de un fichero ejecutable, hay muchas funciones compiladas que pueden ser invocadas por el programa en cuestión; pero un programa también puede invocar funciones compiladas contenidas en un fichero externo. Este fichero externo es una DLL.

Un programa, para acceder a una de las funciones contenidas en una DLL necesita cargarla primero, es decir, solicitar al sistema operativo que la ubique en la memoria del ordenador. Esto no tiene que ocurrir necesariamente cuando se ejecuta el programa, sino únicamente cuando se necesita por primera vez una de sus funciones. Es común, en ocasiones, detectar cierta demora en un programa. A veces, dicha demora viene acompañada de cierta actividad en el disco duro. Es probable que eso se deba a que el programa está cargando la DLL en cuestión. En tales casos, a no ser que el programa descargue explícitamente la DLL, esta demora no ocurrirá en accesos sucesivos a la misma.

También es posible que varios programas distintos puedan invocar una misma DLL. Todo lo que se necesita saber son las funciones que contiene dicha DLL, cómo interactuar con ellas (introducción de parámetros o valores a devolver) y dónde se ubica dentro del árbol de ficheros del disco duro.

Por lo tanto, gracias a las DLLs los ficheros ejecutables pueden ser más pequeños y los programas más eficientes. El que varios de ellos puedan, en teoría, compartir DLLs comunes, permite además un ahorro significativo de espacio en el disco duro.

El entorno R en Windows, aparte del fichero Rgui.exe que es con el que arranca, incorpora varias DLLs adicionales que pueden invocarse en un momento dado. Además, sus diversos módulos (bibliotecas o librerías) pueden incluir sus propias DLLs. Finalmente, R brinda la posibilidad al usuario de crear e incorporar sus propias DLLs, cargarlas y descargarlas a voluntad y acceder a las funciones que contienen mediante un conjunto específico de funciones.

Todos los compiladores para Windows incorporan la posibilidad de crear DLLs. Los detalles varían de unos a otros y considerarlos en su conjunto entrañaría algunas dificultades técnicas. Por ese motivo, se explicará únicamente el funcionamiento del compilador para C de GNU para Windows conocido como MinGW [14], [15].

Supóngase que se ha creado un fichero que contiene una serie de funciones escritas en C que se desean invocar desde R. Basta decir que el comando mínimo necesario para crear una DLL utilizando el compilador MinGW que hay que introducir en la ventana de MS-DOS es:

```
gcc -shared -o archivo.dll archivo.c
```

Tanto gcc, que está contenido en el directorio bin de donde quiera que se haya instalado MinGW, como el fichero en C, han de estar visibles; o en el directorio en el que se ha tecleado el comando; o dentro del path.

Estaría de más indicar que gcc admite comandos adicionales que pueden ser de interés en determinadas circunstancias. La opción `-shared` especifica que se está creando una DLL y `-o` indica que lo que le sigue es el nombre que se le ha querido dar al archivo DLL.

Supóngase que se ha creado el archivo DLL de acuerdo con el procedimiento anterior. Para poder acceder a las funciones que contiene, R tiene que cargar en primera instancia la DLL. Esto se consigue mediante la función `dyn.load()`. Suponiendo que el archivo DLL esté contenida en el directorio `C:/MisDLLs`, bastaría teclear para cargarla:

```
dyn.load("C:/MisDLLs/archivo.dll")
```

Una vez que deja de ser necesaria, se puede descargar tecleando:

```
dyn.unload("C:/MisDLLs/archivo.dll")
```

Una vez cargado el archivo DLL, si éste contiene una función denominada `func1`, se puede comprobar si en efecto está disponible tecleando:

```
is.loaded("func1")
```

que debería ser TRUE de haberse seguido los pasos anteriores. Si el archivo DLL ha sido compilado por un compilador distinto a MinGW, es bastante posible que `is.loaded("func1")` resulte ser `FALSE` porque muchos de ellos tienden a decorar el nombre de las funciones de las DLLs de una manera un tanto impredecible. R no posee ningún comando capaz de enumerar las funciones accesibles dentro de una DLL.

Finalmente, para invocar la función `func1` del archivo DLL, R utiliza la función `.C` (nombre que subraya el hecho de que sólo es válida para funciones escritas en C) que implementa la interfaz requerida entre R y la DLL. Dicho interfaz determina por una parte, la sintaxis de la función `.C` e impone ciertas restricciones en la naturaleza de las funciones de la DLL. Los aspectos fundamentales a tener en cuenta son los siguientes:

- Cómo invoca `.C` a la función y cómo le transfiere datos.
- En particular, cómo asegura `.C` la compatibilidad del tipo de los datos transferidos.
- Cómo devuelve la función los resultados a `.C`.
- Cómo se accede desde R a los datos que la función deposita en `.C` [16].

Para lograr una transferencia de archivos desde R a nuestra función en C, será necesario tener en cuenta la siguiente tabla de conversión de tipo de datos (ver Tabla 7) [17].

Tabla 7. Comprobación y conversión para cada tipo de dato en R.

Tipo de dato	Comprobación	Conversión
Array	`is.array()`	`as.array()`
Carácter	`is.character()`	`as.character()`
Complejo	`is.complex()`	`as.complex()`
Marco de datos	`is.data.frame()`	`as.data.frame()`
Double	`is.double()`	`as.double()`
Factor	`is.factor()`	`as.factor()`
Lista	`is.list()`	`as.list()`
Lógico	`is.logical()`	`as.logical()`
Numérico	`is.numeric()`	`as.numeric()`
Raw	`is.raw()`	`as.raw()`
Series temporales	`is.ts()`	`as.ts()`
Vector	`is.vector()`	`as.vector()`

R señala los resultados inesperados a través de advertencias y errores. Las advertencias se producen cuando el cálculo da un resultado inusual que, sin embargo, no excluye una evaluación adicional. Las advertencias y los errores que se producen en el 'prompt' son generalmente fáciles de diagnosticar. Pueden ser más enigmáticos cuando se producen en una función, exacerbada a veces por mensajes de error crípticos.

Un primer paso para llegar a un acuerdo con los errores es simplificar el problema lo máximo posible, apuntando a un error 'reproducible'. El error reproducible podría incluir un pequeño conjunto de datos (incluso trivial) que provoca inmediatamente el error. A menudo, el proceso de crear un ejemplo reproducible ayuda a aclarar cuál es el error y qué posibles soluciones podría haber. La invocación de `traceback()` inmediatamente después de que ocurra un error, proporciona una 'pila' de las llamadas a funciones que estaban en vigor cuando se produjo el error. Esto puede ayudar a entender el contexto en el que se produjo el error.

Conociendo el contexto, se podría utilizar `debug` para entrar en un navegador (ver `?browser`) que permite dar un paso a través de la función en la que se produjo el error.

A veces puede ser útil el uso de opciones globales (ver `?options`) para influir en lo que ocurre cuando se produce un error. Dos opciones globales comunes son `error` y `warn`. Configurando `error=recover` combina la funcionalidad de `traceback` y `debug`, lo que permite al usuario entrar al navegador en cualquier nivel de la pila de llamadas en vigor en el momento en el que ocurrió el error. El comportamiento por defecto del error puede restaurarse con `options(error=NULL)`. Configurando `warn=2` causaría advertencias al ser promovido a errores.

Por ejemplo, la investigación inicial de un error puede demostrar que el error se produce cuando uno de los argumentos de una función tiene un valor `NaN`. El error puede estar acompañado por un mensaje de advertencia que `NaN` ha introducido, pero debido a que las advertencias por defecto no informan inmediatamente, no está claro de dónde viene `NaN`. `warn=2` significa que el aviso es tratado como un error, y por lo tanto se puede depurar utilizando `traceback, debug, etc.` Existen funciones de depuración útiles adicionales como `browser, trace` y `setBreakpoint`.

3.2. Bioconductor

Bioconductor consiste en una colección de paquetes de R para el análisis y comprensión de datos genómicos de alto rendimiento. Bioconductor se inició hace más de 10 años, ganándose la credibilidad gracias a su rigurosa aproximación estadística a los pre-procesamiento de micro-arrays, al análisis de experimentos diseñados y a la aproximación integradora y reproducible de tareas bioinformáticas. En la actualidad hay más de 600 paquetes de Bioconductor orientados a expresiones y otros microarrays, análisis de secuencias, flujos citométricos, imágenes y otros dominios. La página web de Bioconductor proporciona la instalación, repositorio de paquetes, ayuda, y otra documentación.

Las características que incluye son las siguientes:

- Introducción a flujos de trabajo.
- Un manifiesto de los paquetes de Bioconductor dispuestos en BiocViews.
- Anotación (bases de datos de información genómica relevante) y paquetes con datos experimentales (que contiene conjuntos de datos relativamente completos y sus análisis).
- Listas de correo, incluyendo búsquedas en archivos, como principal fuente de ayuda.
- Información sobre cursos y conferencias, incluyendo un amplio material de referencia.
- Información general sobre el proyecto.
- Recursos del paquete desarrollado, incluidas las directrices para la creación y presentación de nuevos paquetes.

3.2.1. Programación estadística

Existen muchas aplicaciones de software tanto académicas como comerciales. Es por ello que debemos hacernos la pregunta de por qué deberíamos utilizar R y Bioconductor. Una respuesta sería la necesidad y la existencia de herramientas que traten datos genómicos de alto rendimiento, y que a su vez, sean efectivas en aplicaciones de software biológicas.

Software efectivo en biología computacional: Los problemas relacionados con el alto rendimiento, nos hacen utilizar grandes conjuntos de datos. Esto se aplica tanto a datos primarios (valores de expresión de microarrays, lecturas de secuencias, etc.) como a las anotaciones de esos datos (combinaciones de genes y características tales como exones o regiones reguladoras; participación en las secuencias biológicas, etc.). Grandes conjuntos de datos demandan sobre nuestras herramientas que excluyan algunas prácticas habituales, tales como hojas de cálculo. Del mismo modo, las relaciones complejas entre los datos y la anotación, y la diversidad de cuestiones en investigación, requieren la flexibilidad típica de un lenguaje de programación en lugar de una interfaz gráfica limitada de usuario.

El análisis de datos de alto rendimiento es necesariamente estadístico. El volumen de datos requiere que se resuma adecuadamente antes de que sea posible cualquier tipo de

comprensión. Los datos son producidos por tecnologías avanzadas, y éstas introducen artefactos (ej. el sondeo específico sesgado en microarrays; secuencia o base llamando al sesgo en experimentos de RNA-seq) que tienen que ser acomodados con el fin de evitar la inferencia incorrecta o ineficiente. Los conjuntos de datos normalmente derivan de experimentos diseñados que requieren un enfoque estadístico, tanto para tener en cuenta el diseño y abordar correctamente el gran número de valores observados (ej. la expresión de genes o contadores de etiquetas en secuencias) como para un pequeño número de muestras accesibles en experimentos típicos.

La investigación debe ser reproducible. La reproducibilidad es un ideal del método científico y un requerimiento pragmático. Este último viene de la naturaleza a largo plazo y multiparticipativa de la ciencia contemporánea. Un análisis será realizado para el experimento inicial, revisándolo nuevamente durante la preparación del manuscrito y durante las comprobaciones o próximos pasos a determinar. Del mismo modo, los análisis suelen incluir un equipo de personas con diversos ámbitos de competencia. Las colaboraciones resultan efectivas cuando es fácil de reproducir (tal vez con menos modificaciones) un resultado existente, y cuando un sofisticado análisis estadístico o bioinformático pueda ser eficazmente transmitido a otros miembros del equipo.

La ciencia avanza muy rápidamente. Esto se debe a las innovadoras cuestiones que son la clave del descubrimiento, tanto por la innovación tecnológica como por la accesibilidad. La rapidez del desarrollo científico coloca significativamente la carga en el software, que también debe avanzar con rapidez. El software efectivo no puede ser demasiado brillante, ya que eso requeriría que los análisis correctos sean 'conocidos' y que esos recursos significativos de tiempo y dinero se hayan invertido en el desarrollo del software; esto implica que el software se encuentra en la cola de la innovación. Por otra parte, el software de vanguardia no puede ser demasiado idiosincrásico, sino que debe ser utilizable por un público más amplio que el propio creador del software, y acomodarlo con otros programas relevantes para el análisis. En definitiva, el software efectivo debe ser en cierta manera accesible. Otra característica, es la implementación transparente, donde el software innovador esté suficientemente documentado y el código fuente lo suficientemente accesible para las hipótesis, enfoques, las decisiones de

implementación práctica, y los errores de codificación inevitables que deben evaluarse por otros profesionales cualificados. Un aspecto final de la asequibilidad es que el software es en realidad servible. Esto se logra a través de la documentación adecuada, foros de soporte, y oportunidades de capacitación.

Bioconductor como una aplicación software efectiva: A continuación, analizaremos las características de R y Bioconductor que contribuyen a la efectividad como herramienta de software.

Bioconductor es una muy adecuada suite para manejar gran cantidad de datos y anotaciones. Las "clases" de Bioconductor representan datos de alto rendimiento y sus anotaciones de una manera integrada. Los métodos de Bioconductor utilizan técnicas avanzadas de programación o recursos de R (como bases de datos transparentes o acceso a la red) para minimizar los requerimientos de memoria e integrarlo con diversos recursos. Las clases y métodos coordinan conjuntos de datos complejos con una amplia anotación. No obstante, el modelo básico para la manipulación de objetos en R implica vectorizar en memoria las representaciones. Por esta razón, en los paradigmas particulares de programación (ej. procesamiento de secuencias de flujo de datos; paralelismo explícito) o recursos de hardware (ej. ordenadores de gran memoria) son a veces necesarios cuando se trata una gran cantidad de datos.

R es ideal para hacer frente a los desafíos estadísticos de datos de alto rendimiento. Tres ejemplos incluyen el desarrollo de la "RMA" (robusto promedio multiarray) y otros algoritmos de normalización para el pre-procesamiento de microarrays, el uso de t-estadísticas moderadas para la evaluación de microarrays de expresiones diferenciales, y el desarrollo de enfoques binomiales negativos para estimar la dispersión de conteo de lectura necesaria para el análisis apropiado de experimentos RNAseq diseñados.

Muchos de los aspectos antiguos de R y Bioconductor facilitan la investigación reproducible. Un análisis es a menudo representado como un script basado en texto. La reproducibilidad de los análisis implica volver a ejecutar el script; ajustándolo en base a como el análisis esté desempeñado, implicando tareas simples de edición de texto. Más allá de ésto, R

tiene la noción de 'viñeta', que representa un análisis como documento LATEX con comandos R embebidos. La comandos de R se evalúan cuando el documento es construido, reproduciendo así el análisis. El uso de LATEX implica manipulaciones simbólicas en los scripts que aumentan con explicaciones textuales y justificaciones para el enfoque adoptado; que incluyen resúmenes gráficos y tabulares en los lugares apropiados del análisis. R incluye facilidades para informar de la versión exacta de R y paquetes asociados utilizados en el análisis para que, si fuese necesario, las discrepancias entre las versiones de software puedan ser localizadas, y su importancia evaluada. Mientras que los usuarios a menudo piensan sobre los paquetes de R como forma de proporcionar una nueva funcionalidad, los paquetes también pueden ser utilizados para mejorar la reproducibilidad mediante la encapsulación de un único análisis. El paquete puede contener conjuntos de datos, viñetas describiendo análisis, funciones de R que podrían haber sido escritas, scripts para etapas de procesamiento de datos clave, y la documentación (a través de los mecanismos de ayuda estándar R) sobre funciones, datos y paquetes.

El proyecto Bioconductor adopta prácticas que facilitan la reproducibilidad. Las versiones de R y Bioconductor se publican dos veces al año, siendo cada versión de Bioconductor el resultado de un desarrollo, de una rama separada, durante los seis meses anteriores. La liberación es construida todos los días sobre la versión correspondiente de R para Linux, Mac y Windows, con un amplio conjunto de pruebas realizadas. La función `biocLite()` asegura que cada versión de R utiliza los paquetes de Bioconductor correspondientes. Así pues, el usuario tiene acceso a versiones de los paquetes probados y estables, siendo de esta manera R y Bioconductor, herramientas efectivas para la investigación reproducible.

R y Bioconductor es justo y equilibrado en términos de accesibilidad. El software está disponible gratuitamente y el código fuente es fácil y plenamente accesible para la evaluación crítica. El sistema de empaquetamiento y de verificación de R requiere que todas las funciones estén documentadas. Bioconductor requiere que cada paquete contenga viñetas para ilustrar el uso del software. También dispone de listas de correo activas para R y Bioconductor para el apoyo inmediato, formación regular y actividades de conferencia para el desarrollo profesional.

3.2.2. Bioconductor en el análisis de secuencias de alto rendimiento

La tabla inferior, enumera muchos de los paquetes disponibles para el análisis de secuencias. La tabla incluye paquetes para la representación de datos de secuencias relacionadas (ej. GenomicRanges, Biostrings), así como el análisis de dominio específico como RNA-seq (ej. edgeR, DEXSeq), ChIP-seq (ej. ChIPpeakAnno, DiffBind), y SNPs y la variación de copias numéricas (ej. genoset, ggtools, VariantAnnotation).

A continuación, se muestra una selección de paquetes de Bioconductor para el análisis de secuencias de alto rendimiento (Tabla 8) [18]:

Tabla 8. Selección de paquetes de Bioconductor para el análisis de secuencias de alto rendimiento.

Concepto	Significado	Paquetes
Representación de datos	Representación de secuencias, de rangos, genómicas,...	IRanges , GenomicRanges, GenomicFeatures, Biostrings, BSgenome, girafe.
Entrada / salida	Entrada y salida en R de datos genómicos	ShortRead (fastq), Rsamtools (bam), rtracklayer (gff, wig, bed), VariantAnnotation (vcf), R453Plus1Toolbox.
Anotaciones	Mapeo y respresentación de anotaciones	GenomicFeatures, ChIPpeakAnno, VariantAnnotation.
Alineamiento	Secuenciación	Rsubread, Biostrings.
Visualización	Gráficos	ggbio, Gviz.
Estimación de calidad	Control de calidad	qrqc , seqbias, ReQON , htSeqTools, TEQC, Rolexa, ShortRead.
RNA-seq	Secuencias de datos ARN	BitSeq, cqn, cummeRbund, DESeq, DEXSeq, EDASeq, edgeR, gage , goseq, iASeq, tweeDEseq.
ChIP-seq	Interacción	BayesPeak, baySeq, ChIPpeakAnno, chipseq,

	proteína-ADN	ChIPseqR, ChIPsim, CSAR, DiffBind, MEDIPS, mosaics, NarrowPeaks, nucleR, PICS, PING, REDseq, Repitools, TSSi.
Patrones	Descubrimiento de patrones	BCRANK , cosmo , cosmoGUI , MotIV , seqLogo , rGADEM.
3C, etc.	Captura de conformación cromosómica	HiTC, r3Cseq.
Copias numéricas	Procesamiento de CNVs y CNAs	cn.mops, CNAnorm, exomeCopy , seqgmentSeq.
Microbioma	Manipulación de datos para microbiomas	Phyloseq, DirichletMultinomial, clstutils, manta, mcaGUI.
Flujos de trabajo	Análisis de datos genómicos de alto rendimiento	ArrayExpressHTS, Genominator, easyRNASeq, oneChannelGUI , rnaSeqMap
Base de datos	Repositorios de datos secuenciados	SRAdb.

4. Análisis de secuencias

En este capítulo se realizará una introducción a las propiedades biológicas fundamentales de las macromoléculas y a su representación terminológica en forma de código para el tratamiento de datos biológicos en el análisis de secuencias.

La secuenciación de la insulina por el premio Nobel Alfred Sanger, inauguraba una era moderna para la biología molecular y estructural. Por aquel entonces, la biología comenzaba a conocer sus primeros fundamentos en relación al conjunto de datos: las secuencias moleculares. A principios de 1960, se fueron acumulando lentamente las secuencias de las proteínas conocidas ya que, hasta entonces, nadie había manipulado y analizado cadenas moleculares como textos. La mayoría de los métodos tuvieron que ser inventados desde cero, y durante el proceso, una nueva área de investigación (el análisis de secuencias utilizando ordenadores) fue generada. Este fue el origen de la bioinformática.

4.1. Secuencias de proteínas

Las proteínas abundan en la carne, en el pescado y en la verdura. Además, todas estas proteínas se componen de los mismos bloques de construcción básicos que se conocen con el nombre de aminoácidos. Los aminoácidos son moléculas orgánicas bastante complejas, formadas por átomos de carbono, hidrógeno, oxígeno, nitrógeno, y azufre.

Los bioquímicos se dieron cuenta conforme avanzaba el tiempo, que las proteínas eran grandes moléculas (macromoléculas) compuestas por un gran número de aminoácidos (típicamente entre 100 y 500) donde hicieron una selección de veinte (ver Tabla 9).

Tabla 9. Los 20 aminoácidos junto con sus códigos oficiales.

#	Código 1 Letra	Código 3 Letras	Nombre
1	A	Ala	Alanina
2	R	Arg	Arginina
3	N	Asn	Asparagina
4	D	Asp	Ácido aspártico
5	C	Cys	Cisteína
6	Q	Gln	Glutamina
7	E	Glu	Ácido glutámico
8	G	Gly	Glicina
9	H	His	Histidina
10	I	Ile	Isoleucina
11	L	Leu	Leucina
12	K	Lys	Lisina
13	M	Met	Metionina
14	F	Phe	Fenilalanina
15	P	Pro	Prolina
16	S	Ser	Serina
17	T	Thr	Treonina
18	W	Trp	Triptófano
19	Y	Tyr	Tirosina
20	V	Val	Valina

Cuando se trabaja con bases de datos o aplicaciones de análisis, es muy probable que aparezcan letras inusuales de vez en cuando en las secuencias de proteínas. Estas letras se utilizan, ya sea para designar a los aminoácidos excepcionales, como para referirse a varios niveles de ambigüedad (ver Tabla 10).

Tabla 10. Los 7 códigos ambiguos o excepcionales de los aminoácidos.

Código 1 Letra	Código 3 Letras	Nombre
B	Asn o Asp	Asparragina o ácido aspártico
J	Xle	Isoleucina o leucina
O	Pyl	Pirrolisina
U	Sec	Selenocisteína
Z	Gln o Glu	Glutamina o ácido glutámico
X	Xaa	Cualquier residuo
-	---	No corresponde a ningún residuo (gap)

4.2. Secuencias de ADN

El ADN (ácido desoxirribonucléico) es el miembro más digno de la familia de los ácidos nucleicos de las macromoléculas. Su única tarea es garantizar la conservación de la información genética de su organismo. Por lo tanto, es muy estable y resistente y se encuentra bien protegido en el núcleo de cada célula. Como fue el caso de los 20 aminoácidos encontrados en las proteínas, los cuatro nucleótidos que elaboran el ADN se componen del mismo cuerpo y contienen el mismo par de "ganchos" (ver Tabla 11).

Tabla 11. Los códigos más comunes utilizados en secuencias de nucleótidos de ADN.

Código de 1 letra	Nombre del nucleótido	Categoría
A	Adenina	Purina
C	Citosina	Pirimidina
G	Guanina	Purina
T	Timina	Pirimidina
N	Cualquier nucleótido	(n/a)
R	A o G	Purina
Y	C o T	Pirimidina
-	----	Nada (gap)

Además, la molécula de ADN consiste en 2 hebras complementarias (al igual que sucede con el ARN). Esta última, tiene el impulso natural para emparejarse con secuencias complementarias (ver Figura 2).

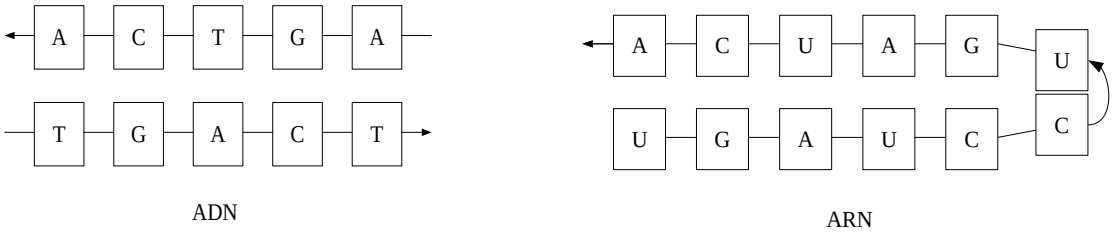


Figura 2. A la izquierda las hebras de una molécula de ADN. A la derecha las hebras de una molécula de ARN intentando emparejarse con secuencias complementarias.

4.3. Secuencias de ARN

El ácido ribonucleico (ARN) es el miembro más activo de la familia de los ácidos nucleicos de las macromoléculas. Es sintetizada y degradada constantemente realizando copias de genes. En el contexto de la bioinformática, solo existen dos diferencias importantes entre el ADN y el ARN:

- El ARN se diferencia del ADN en tan sólo un nucleótido (ver Tabla 12).
- El ARN viene solamente con una simple hebra, no con una hélice [19].

Tabla 12. Los códigos más comunes usados en secuencias de nucleotidos ARN.

Código de 1 Letra	Nucleótido	Categoría
A	Adenine	Purina
C	Cytosine	Pirimidina
G	Guanine	Purina
U	Uracil	Pirimidina
N	Cualquiero nucleotido	Purina o pirimidina
R	A o G	Purina
Y	C o U	Pirimidina
-	-----	Nada (gap)

5. Alineamiento de secuencias por pares

En este capítulo, nos centraremos en uno de los puntos más importantes del análisis bioinformático. La comparación de secuencias, trata de un importante primer paso hacia el análisis estructural y funcional de las secuencias recién determinadas. A medida que nuevas secuencias biológicas se están generando a un ritmo exponencial, la comparación de secuencias es cada vez más importante para hacer inferencia funcional y evolutiva de una nueva proteína con proteínas que ya existen en la base de datos. El proceso más fundamental en este tipo de comparación es el alineamiento de secuencias. Este es el proceso por el cual las secuencias son comparadas mediante la búsqueda de patrones de caracteres comunes y el establecimiento de la correspondencia entre residuos en las secuencias relacionadas. El alineamiento de secuencias por pares es el proceso de alinear dos secuencias y es la base de la búsqueda de similitud en bases de datos y en los alineamientos múltiples de secuencias.

5.1. Base evolutiva

El ADN y las proteínas son productos de la evolución. Los bloques de construcción de estas macromoléculas biológicas, bases de nucleótidos y aminoácidos, forman secuencias lineales que determinan la estructura primaria de las moléculas. Estas moléculas pueden ser consideradas fósiles moleculares que codifican la historia de millones de años de evolución. Durante este período de tiempo, las secuencias moleculares se someten a cambios al azar, algunas de las cuales se seleccionan durante el proceso de la evolución. Como las secuencias seleccionadas, se acumulan gradualmente mutaciones que divergen con el tiempo; permaneciendo las huellas de la evolución en ciertas porciones de las secuencias y permitiendo todavía la identificación de la ascendencia común. La presencia de restos evolutivos se debe a que algunos de los residuos que desempeñan funciones funcionales y estructurales claves, tienden a ser preservados por la selección natural. Otros residuos que pueden ser menos cruciales para la estructura y la función, tienden a mutar con más frecuencia. Por ejemplo, los residuos del sitio activo de una familia de enzimas tienden a conservarse, ya que son responsables de las funciones catalíticas. Por lo tanto , mediante la comparación de secuencias a través del alineamiento, los patrones de conservación y las variaciones pueden ser identificadas. El grado

de conservación de la secuencia en el alineamiento revela la relación evolutiva de diferentes secuencias, mientras que la variación entre secuencias refleja los cambios que se han producido durante la evolución en forma de sustituciones, inserciones y deleciones.

Identificando las relaciones evolutivas entre secuencias ayuda a caracterizar la función de secuencias desconocidas. Cuando un alineamiento de secuencias revela una similitud significativa entre un grupo de secuencias, pueden ser consideradas como pertenecientes a una misma familia. Si uno de los miembros de la familia tiene una estructura y una función conocida, a continuación, esa información puede ser transferida a las que todavía no se han caracterizado experimentalmente. Por lo tanto, el alineamiento de secuencias se puede utilizar como base para la predicción de la estructura y de la función de secuencias no caracterizadas.

El alineamiento de secuencias proporciona inferencia para la relación de dos secuencias en estudio. Si las dos secuencias comparten una similitud significativa, es muy poco probable que la amplia similitud entre las dos secuencias se haya adquirido al azar, lo que significa que las dos secuencias deben haber derivado de un origen evolutivo común. Cuando un alineamiento de secuencias se genera correctamente, refleja la relación evolutiva de las dos secuencias: las regiones que están alineadas pero que no son idénticas, representan sustitución de residuos; las regiones donde los residuos de una secuencia no se corresponden en nada con la otra, representan inserciones o deleciones que se han producido en una de las secuencias durante la evolución. También es posible que dos secuencias hayan derivado de un ancestro común, pero que hayan podido divergir de tal manera que las relaciones ancestrales comunes no sean reconocibles a nivel de secuencia.

5.2. Secuencias homólogas frente a similitud de secuencias

Un concepto importante en el análisis de secuencias, es el de homología de secuencias. Cuando dos secuencias son descendientes de un origen evolutivo común, se dice que tienen una relación homologa o que comparten homología. Un término relacionado pero diferente es la similitud de secuencia, que es el porcentaje de residuos alineados que son similares en propiedades físico-químicas tales como tamaño, carga e hidrofobicidad.

Es importante distinguir la homología de secuencia del término relacionado conocido como similitud de secuencia, porque los dos términos son confundidos a menudo por algunos investigadores que los utilizan indistintamente en la literatura científica. Para ser claros, la homología de secuencia es una inferencia o conclusión de una relación ancestral común extraída de la comparación de similitud de secuencias cuando las dos secuencias comparten un grado suficiente de similitud. Por otra parte, la similitud es un resultado directo de la observación del alineamiento de secuencia. La similitud de secuencia se puede cuantificar utilizando porcentajes; la homología, en cambio, es una declaración cualitativa. Por ejemplo, uno puede decir que dos secuencias comparten 40% de similitud. Es incorrecto decir que las dos secuencias comparten el 40% de homología, ya sean homólogas o no.

En general, si el nivel de similitud de secuencia es lo suficientemente alta, la relación evolutiva común puede deducirse. Al tratar con problemas reales en investigación, la cuestión de en qué nivel de similitud puede uno inferir relaciones homólogas no siempre es clara. La respuesta depende del tipo de secuencias que se estén examinando y de la longitud de las mismas. Las secuencias de nucleótidos consisten en sólo cuatro caracteres, y por lo tanto, las secuencias no relacionadas tienen por lo menos un 25% de probabilidades de ser idénticas. Para las secuencias de proteínas, hay veinte residuos de aminoácidos posibles, por lo que dos secuencias no relacionadas pueden coincidir al menos en el 5% de los residuos al azar. Si los huecos (gaps) son permitidos, el porcentaje podría aumentar al 10-20%, sin olvidar que la longitud de secuencia es también un factor crucial. Cuanto más corta sea la secuencia, mayor será la probabilidad de que algunos de los alineamientos sea atribuible al azar. Cuanto más larga sea la secuencia, menos probable será la coincidencia en el mismo nivel de similitud atribuible al azar.

Esto sugiere que las secuencias más cortas requieren puntos de corte más altos para inferir relaciones homólogas que las secuencias más largas. Para determinar una relación de homología de dos secuencias de proteínas (ver Figura 3), por ejemplo, si las dos secuencias están alineadas en toda su longitud, que son 100 residuos de longitud, una identidad del 30% o superior puede considerarse con seguridad que tienen una estrecha homología. A veces se refieren a ello como "zona segura". Si su nivel de identidad cae entre un 20% y 30%, la

determinación de las relaciones homólogas en esta gama se hace menos cierta. Esta es la zona a menudo considerada como la "zona gris", donde homólogos remotos se mezclan con secuencias relacionadas al azar. Por debajo del 20% de identidad, donde una gran proporción de las secuencias no relacionadas están presentes, las relaciones homólogas no se pueden determinar de forma fiable y por lo tanto entran en la "zona de medianoche." Es necesario destacar que los valores de identidad porcentuales sólo proporcionan una guía provisional para la identificación de homología, por lo tanto, no es una regla precisa para determinar las relaciones de secuencias, especialmente para las secuencias en la “zona gris”.

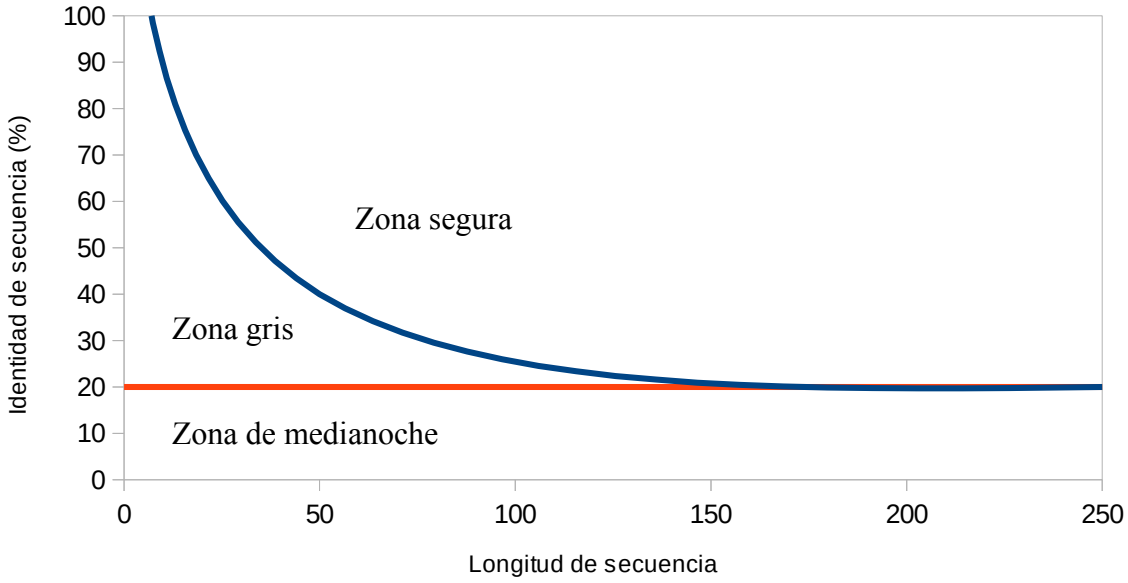


Figura 3. Las tres zonas de los alineamientos de proteínas.

5.3. Similitud de secuencias frente a secuencias identidad

Otro conjunto de términos relacionados con la comparación de secuencias son la similitud de secuencia y la identidad de secuencia. La similitud de secuencia y la identidad de secuencia son sinónimos para secuencias de nucleótidos. Para las secuencias de proteínas, sin embargo, los dos conceptos son muy diferentes. En el alineamiento de secuencias en proteínas, la identidad de secuencia se refiere al porcentaje de emparejamientos de los mismos residuos de aminoácidos entre dos secuencias alineadas. La similitud se refiere al porcentaje de residuos alineados con características físico-químicas similares y que pueden ser más fácilmente sustituibles entre sí.

Hay dos maneras de calcular la similitud/identidad de secuencia. Una de ellas implica el uso de toda las longitud de las dos secuencias, mientras que la otra se normaliza por el tamaño de la secuencia más corta. El primer método utiliza la siguiente fórmula:

$$S = [(L_s \times 2)/(L_a + L_b)] \times 100 \qquad (20)$$

donde S es el porcentaje de similitud de secuencia, L_s es el número de residuos alineados con características similares, y L_a y L_b son las longitudes totales de cada secuencia individual. La identidad de secuencia (I%) se puede calcular de una manera similar, donde L_i es el número de residuos idénticos alineados:

$$I = [(L_i \times 2)/(L_a + L_b)] \times 100 \qquad (20)$$

El segundo método de cálculo es para derivar el porcentaje de residuos idénticos/similares sobre la longitud completa de la secuencia más pequeña utilizando la siguiente fórmula, donde L_a es la longitud de la más corta de las dos secuencias:

$$I(S)\% = L_{i(s)}/L_a\% \qquad (20)$$

5.4. Programación dinámica

La programación dinámica es un método para la resolución de problemas complejos desglosándolos en subproblemas más simples. Es aplicable a los problemas que presentan propiedades de subproblemas superpuestos [21] y subestructuras óptimas. Cuando es aplicable, el método toma mucho menos tiempo que los métodos ingenuos, que no se aprovechan de la ventaja de los subproblemas superpuestos (como la búsqueda en profundidad).

La idea detrás de la programación dinámica es muy simple. En general, para resolver un problema dado, tenemos que resolver diferentes partes del problema (subproblemas) para, a continuación, combinar las soluciones de los subproblemas y llegar a una solución global. A menudo, cuando se utiliza un método más ingenuo, muchos de los subproblemas se generan y se resuelven muchas veces. El enfoque de la programación dinámica busca resolver cada subproblema sólo una vez, lo que reduce el número de cálculos. Una vez que la solución a un subproblema dado se ha calculado, se almacena o "memoiza" para la próxima vez que se necesite la misma solución. Este enfoque es especialmente útil cuando el número de subproblemas que se repiten crece exponencialmente como una función del tamaño de la entrada.

Los algoritmos de programación dinámica son utilizados para la optimización (ej. encontrar el camino más corto entre dos puntos, o la manera más rápida de multiplicar varias matrices). Un algoritmo de programación dinámica examinará todas las formas posibles de resolver un problema y elegirá la mejor solución. Por lo tanto, de la programación dinámica podemos pensar en términos generales como un método inteligente de fuerza bruta que nos permite ir a través de todas las soluciones posibles, para elegir la mejor. Si la magnitud del problema es tal que es factible pasar por todas las soluciones posibles y lo suficientemente rápido, la programación dinámica garantiza encontrar la solución óptima. Las alternativas son muchas, como el uso de un algoritmo voraz, que recoge la mejor opción posible "en cualquier posible ramificación del camino" es más rápido, pero no garantiza la solución óptima. Afortunadamente, algunos algoritmos voraces (como los árboles de expansión mínimos) han demostrado conducir a la solución óptima.

Por ejemplo, digamos que tenemos que ir del punto A al punto B lo más rápido posible en una determinada ciudad durante las horas punta. Un algoritmo de programación dinámica verá el informe de tráfico, mirando todas las combinaciones posibles de las vías que podamos tomar; y sólo entonces nos dirá qué camino es el más rápido. Por supuesto, es posible que tengamos que esperar un tiempo hasta que termine el algoritmo para que podamos empezar a conducir. El camino que tomaremos será el más rápido (suponiendo que nada cambie en el ambiente externo). Por otro lado, un algoritmo voraz comenzará inmediatamente a conducirnos, cogiendo la carretera que le parezca más rápida en cada intersección. Como podemos imaginar, esta estrategia no podría darnos la hora de llegada más rápida, ya que podría tomar algunas calles 'fáciles' y luego encontrarnos atrapados irremediablemente en un atasco de tráfico.

A veces, aplicar la memoización a una solución recursiva básica ingenua se traduce en una solución de programación dinámica óptima. Sin embargo, muchos problemas requieren algoritmos de programación dinámica más sofisticados. Algunos de éstos pueden también ser recursivos, pero con diferente parametrizado que la solución ingenua. Otros pueden ser más complicados y no pueden ser implementados como una función recursiva con memoización. Ejemplos de estos casos son las dos soluciones al rompecabezas de la caída del huevo explicado más abajo.

La programación dinámica es a la vez un método de optimización matemática y un método de programación de computadoras. En ambos contextos se refiere a la simplificación de un problema complejo dividiéndolo en subproblemas más simples de manera recursiva. Mientras que algunos problemas de decisión no se pueden separar de esta manera, las decisiones que abarcan varios puntos en el tiempo con frecuencia se descomponen recurrentemente. Bellman llamó a ésto "Principio de optimalidad". Asimismo, en ciencias de la computación, se dice que un problema que se puede descomponer recursivamente, tiene una subestructura óptima.

Si los subproblemas se pueden anidar recursivamente dentro de problemas más grandes, se pueden aplicar los métodos de programación dinámica, habiendo una relación entre el valor de un problema mayor y los valores de los subproblemas [22]. En la literatura, a esta relación se la conoce como la ecuación de Bellman.

Hay dos atributos clave que un problema debe tener para que la programación dinámica le sea aplicable: subestructura óptima y subproblemas superpuestos. Si un problema puede ser resuelto mediante la combinación de soluciones óptimas a subproblemas no superpuestos, en su lugar, la estrategia se llamaría "divide y vencerás". Esta es la razón por la que el ordenamiento por mezcla (mergesort) y el ordenamiento rápido (quicksort) no están clasificados como problemas de programación dinámica.

Una subestructura óptima significa que la solución dada a un problema de optimización, puede ser obtenida mediante la combinación de soluciones óptimas a sus subproblemas. Por consiguiente, el primer paso hacia la elaboración de una solución de programación dinámica es comprobar si en el problema exhibe tal subestructura óptima. Tal subestructura óptima se describe generalmente por medio de la recursión. Por ejemplo, dado un grafo $G = (V, E)$, el camino más corto 'p' desde un vértice 'u' a un vértice 'v' presenta una subestructura óptima; tomando cualquier vértice intermedio 'w' en el camino más corto 'p'. Si 'p' es verdaderamente el camino más corto, entonces se podrá dividir en subtrazados p_1 de 'u' a 'w' y p_2 de 'w' de 'v' tal que éstos, a su vez, son de hecho los caminos más cortos entre los vértices correspondientes. Por lo tanto, uno puede formular fácilmente la solución para encontrar los caminos más cortos de manera recursiva, que es lo que el algoritmo de Bellman Ford o el algoritmo de Floyd Warshall hacen.

Superponer subproblemas significa que el espacio de los subproblemas debe de ser pequeño, es decir, que cualquier algoritmo recursivo que resuelva el problema debe resolver los mismos subproblemas una y otra vez, en lugar de generar nuevos subproblemas. La programación dinámica tiene en cuenta este hecho y resuelve cada subproblema una sola vez.

Ésto se puede lograr de varias maneras:

- Enfoque top-down (de arriba abajo): Si la solución a cualquier problema se puede formular de forma recursiva utilizando la solución a sus subproblemas, y si sus subproblemas se encuentran superpuestos, entonces uno puede fácilmente memoizar o almacenar las soluciones de los subproblemas en una tabla. Siempre que se intente resolver un nuevo subproblema, primero revisaremos la tabla para ver si ya está resuelto. Si se ha guardado una solución, se puede utilizar directamente, si no, se resuelve el subproblema y se añade la solución a la tabla.

- Enfoque bottom-up (de abajo arriba): Una vez formulada la solución a un problema de forma recursiva como en términos de sus subproblemas, podemos tratar de reformular el problema de una manera ascendente: intentar resolver los subproblemas primero y utilizar sus soluciones para construir y llegar a las soluciones de los subproblemas más grandes. Ésto también se hace generalmente en forma de tabla mediante la generación iterativa de soluciones a los subproblemas cada vez más grandes mediante el uso de las soluciones de los subproblemas más pequeños.

Lo siguiente, es una descripción del famoso rompecabezas que involucra $n=2$ huevos y un edificio de $h=36$ pisos [23]. Supóngase que queremos saber qué plantas de un edificio de 36 plantas son seguras para dejar caer los huevos, y cuáles harán que los huevos se rompan (utilizando la terminología inglesa de EE.UU., en donde la primera planta se encuentra a nivel del suelo). Hacemos algunas suposiciones:

- Un huevo que sobrevive a una caída puede ser utilizado de nuevo.
- Un huevo roto debe ser desechado.
- El efecto de una caída es el mismo para todos los huevos.
- Si un huevo se rompe cuando se cae, entonces se rompería si se deja caer desde una ventana superior.
- Si un huevo sobrevive a una caída, entonces sobreviviría a una caída más corta.

- No se descarta que las ventanas del primer piso rompa los huevos, al igual que tampoco es descartable que las ventanas del piso 36 no quiebren los huevos.

Si sólo hay un huevo disponible y queremos estar seguros de obtener el resultado correcto, el experimento se puede realizar de una sola manera. Se suelta el huevo desde la ventana del primer piso, y si sobrevive, se deja caer desde la ventana del segundo piso. Se continua subiendo hasta que se rompa. En el peor de los casos, este método puede requerir 36 caídas. Supongamos que hay 2 huevos disponibles. ¿Cuál es el menor número de caídas que garantiza que funcione en todos los casos?

Para obtener una ecuación funcional de programación dinámica para este rompecabezas, se deja que el estado del modelo de programación dinámica sea un par $s = (n, k)$, donde

n = número de huevos de prueba disponibles, $n = 0, 1, 2, 3, ..., N - 1$.
k = número de pisos (consecutivos) todavía por probar, $k = 0, 1, 2, ..., H – 1$.

Para la instancia, $s = (2,6)$ indica que dos huevos de prueba están disponibles y 6 plantas (consecutivas) todavía no se han probado. El estado inicial del proceso es $s = (N, H)$, donde N indica el número de huevos disponibles al comienzo del experimento. El proceso termina cuando no haya más huevos de prueba ($n = 0$) o cuando $k = 0$, lo que primero ocurra. Si la terminación se produce en el estado $s = (0, k)$ y $k > 0$, entonces la prueba ha fallado. Ahora, teniendo:

$W(n, k)$ = número mínimo de intentos requeridos para identificar el valor de la planta crítica en el peor de los escenarios, dado que el proceso está en el estado $s = (n, k)$. Entonces se puede demostrar que [24]:

$$W(n, k) = 1 + \min \{\max (W (n - 1, x - 1), W(n, k - x)): x = 1, 2, ..., k\}$$

con $W(n, 1) = 1$ para todo $n>0$ y $W(1, k) = k$ para todo k. Es fácil de resolver esta ecuación iterativa mediante el aumento sistemático de los valores de n y k.

5.5. Métodos

El objetivo general del alineamiento de secuencias por pares es encontrar el mejor emparejamiento de dos secuencias, de manera que exista una correspondencia máxima entre los residuos. Para lograr este objetivo, una secuencia necesita ser desplazada con respecto a la otra para encontrar la posición en la que se encuentren las coincidencias máximas. Hay dos estrategias diferentes de alineamiento que se utilizan con frecuencia: el alineamiento global y el alineamiento local.

5.5.1. Alineamiento global y local

En el alineamiento global, dos secuencias que van a ser alineadas se supone que son generalmente similares en toda su longitud. El alineamiento se lleva a cabo desde el principio hasta el final de ambas secuencias para encontrar el mejor alineamiento posible a través de la longitud total entre las dos secuencias. Este método es más aplicable para alinear dos secuencias estrechamente relacionadas de aproximadamente la misma longitud. Para las secuencias divergentes y secuencias de longitudes variables, este método puede no ser capaz de generar resultados óptimos, ya que no reconoce las regiones locales muy similares entre las dos secuencias.

El alineamiento local, por otra parte, no asume que las dos secuencias en cuestión tengan similitud en toda la longitud. Sólo encuentra regiones locales con el más alto nivel de similitud entre las dos secuencias y alinea estas regiones sin tener en cuenta el alineamiento del resto de las regiones de la secuencia. Este enfoque se puede utilizar para el alineamiento de secuencias más divergentes con el objetivo de buscar patrones conservados en secuencias de ADN o proteínas. Las dos secuencias a alinear pueden ser de diferentes longitudes. Este enfoque es más apropiado para el alineamiento de secuencias biológicas divergentes que contienen sólo los módulos que son similares, que se conocen como *dominios* o *motivos*. En la Tabla 13 se muestra un ejemplo de comparación de secuencias por pares mostrando las diferencias entre alineamientos globales y locales.

Tabla 13. Un ejemplo de comparación de secuencias por pares mostrando las diferencias entre alineamientos globales y locales.

	Alineamiento de secuencias global
Secuencia 1	GTAGGCTTAAGGTTA
Secuencia 2	- TAG - - - - A - - - T - A
	Alineamiento de secuencias local
Secuencia 1	TAG
Secuencia 2	TAG

5.5.2. Algoritmos de alineamiento

Los algoritmos de alineamiento, tanto el global como el local, son fundamentalmente similares y sólo difieren en la estrategia de optimización utilizada en el alineamiento de residuos similares. Ambos tipos de algoritmos pueden basarse en algunos de los siguientes tres métodos:

- El método de la matriz de puntos.
- El método de programación dinámica.
- El método heurístico de palabra corta.

Método de matriz por puntos

El método de alineamiento de secuencias más básico es el método de matriz por puntos, también conocido como el método gráfico por puntos (dotplot). Es una manera gráfica de comparar dos secuencias en una matriz de dos dimensiones. En una matriz de puntos, las dos secuencias a comparar son escritas en el eje horizontal y vertical de la matriz. La comparación se realiza mediante el escaneo de cada residuo de una secuencia por similitud con todos los residuos en la otra secuencia. Si una coincidencia residual es encontrada, un punto es colocado dentro del gráfico. De lo contrario, las posiciones de la matriz serían dejadas en blanco.

Cuando dos secuencias tienen regiones sustanciales de similitud, se forma una fila de puntos para crear lineas contiguas y diagonales que revelan la secuencia de alineamiento. Si hay interrupciones en medio de una línea en diagonal, indica que hay inserciones o deleciones.

Existe un problema cuando se comparan largas secuencias utilizando el método de la matriz por puntos, concretamente el alto nivel de ruido. En la mayoría de los gráficos por puntos, los puntos son trazados sobre el gráfico, ocultando la identificación del alineamiento verdadero. Para secuencias de ADN, el problema es especialmente grave, ya que sólo hay cuatro caracteres posibles y por lo tanto, cada residuo tiene una posibilidad entre cuatro de igualar un residuo con la otra secuencia. Para reducir el ruido, en vez de utilizar un único residuo para escanear similitudes, se aplica una técnica de filtrado que utiliza una 'ventana' de longitud fija que abarca un tramo de pares de residuos. Al aplicar el filtro, la ventana se desliza sobre las dos secuencias para comparar todos los posibles tramos. Los puntos sólo se colocan cuando un tramo de residuos equivalente al tamaño de la ventana de una secuencia coincide completamente con el tramo de la otra secuencia. Este método ha resultado ser eficaz en la reducción del nivel de ruido.

La ventana también recibe el nombre de tupla (tuple), y es el tamaño con el cual puede ser manipulado de modo que las coincidencias en un patrón de la secuencia pueda ser trazado. Sin embargo, si el tamaño de la ventana seleccionada es demasiado larga, se pierde la sensibilidad del alineamiento.

El método de matriz por puntos proporciona una visión directa entre la relación de dos secuencias y ayuda a identificar fácilmente regiones de mayor similitud. Una ventaja particular de este método es la identificación de regiones de repetición en secuencias basadas en la presencia de diagonales paralelas del mismo tamaño verticalmente u horizontalmente en la matriz. El método tiene, por consiguiente, algunas aplicaciones en el campo de la genómica. Es útil en la identificación de repeticiones cromosómicas y en la conservación de orden de los genes entre dos genomas estrechamente relacionados. También puede ser usado en la identificación de estructuras secundarias de ácidos nucleicos a través de la detección auto-complementaria de una secuencia.

El método de matriz por puntos muestra todas las posibles coincidencias en una secuencia. Sin embargo, a menudo es el usuario el que construye un alineamiento completo con inserciones y borrados uniendo diagonales cercanas. Otra limitación de este método de análisis visual es que carece de rigor estadístico en la evaluación de la calidad del alineamiento. El método también está restringido a alineamientos por pares. Es difícil para el método ampliarse a múltiples alineamientos.

Método de programación dinámica

La programación dinámica es un método que determina el alineamiento óptimo haciendo coincidir dos secuencias para todos los posibles pares de caracteres entre las dos secuencias, basándose en el principio de optimalidad de Bellman. El objetivo básico de la programación dinámica consiste en descomponer un problema de optimización para reducir el tiempo de ejecución de un algoritmo. En este sentido, se podría decir que la programación dinámica se basa también en un método de descomposición. Es fundamentalmente similar al método de matriz por puntos, ya que también crea una matriz de alineamiento de dos dimensiones. Sin embargo, el alineamiento se encuentra de una manera más cuantitativa mediante la conversión de una matriz de puntos en una matriz de puntuación para dar cuenta de las coincidencias y diferencias entre las secuencias. Mediante la búsqueda del conjunto de puntuaciones más altas en esta matriz, se puede obtener el mejor alineamiento con precisión.

A continuación, se muestra de manera pormenorizada cómo se completaría la matriz de puntaje en un algoritmo de alineamiento. La programación dinámica trabaja primero con la construcción de una matriz bidimensional en cuyos ejes están las dos secuencias a comparar. El emparejamiento de un residuo, será de acuerdo a una matriz de puntuación. Las puntuaciones serán calculadas fila por fila. Comienza por la primera fila de una secuencia, que es utilizada para escanear a través de toda la longitud de la otra secuencia, continuando con la exploración de la segunda fila; de esta manera, la puntuación de emparejamientos es calculada. El escaneo de la segunda fila tiene en cuenta los resultados ya obtenidos durante la primera ronda. El mejor resultado, se coloca en la esquina inferior derecha de la matriz intermedia (ver Figura 4).

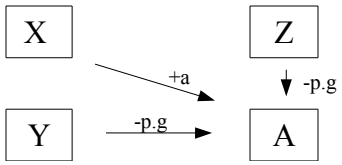
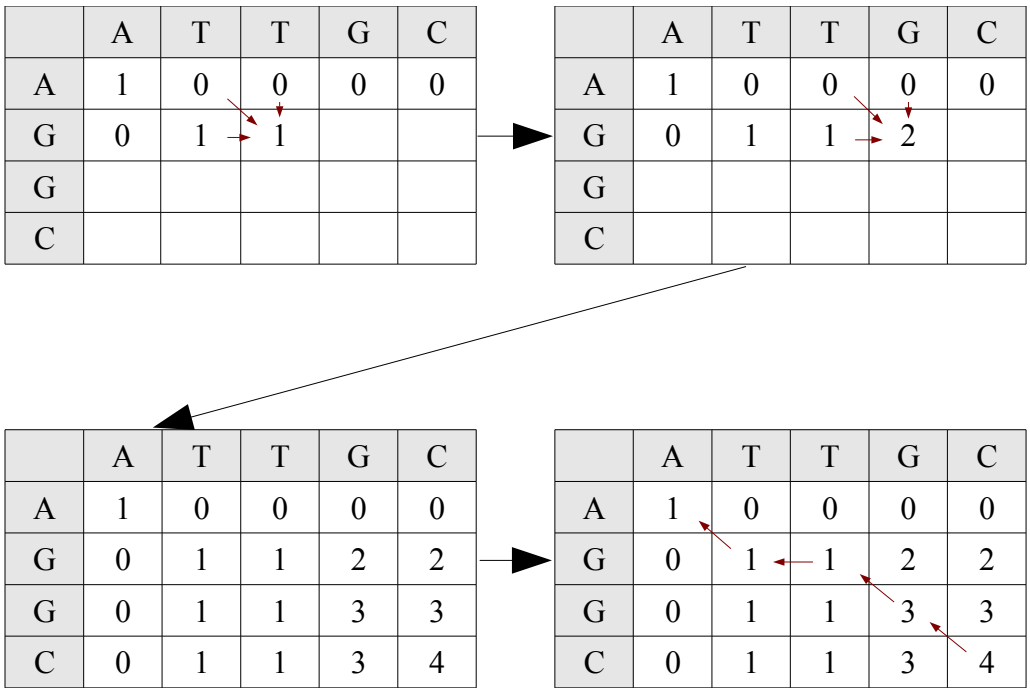


Figura 4. *A* es la puntuación máxima de una de las tres direcciones (*X*,*Y*,*Z*) más y menos la puntuación de coincidencia en la posición actual (*a*) para la esquina inferior derecha y una penalización por hueco (*p.g*) respectivamente.

Este proceso itera hasta que los valores para todas las celdas son asignados. Por lo tanto, las puntuaciones se acumulan a lo largo de la diagonal que va desde la esquina superior izquierda a la esquina inferior derecha. Una vez que los resultados se hayan acumulado en la matriz, el siguiente paso será encontrar el camino que represente el alineamiento óptimo. Esto se consigue trazando a través de la matriz en orden inverso desde la esquina inferior derecha de la matriz (alineamiento global) hacia el origen de la matriz en la esquina superior izquierda. El mejor camino de emparejamiento, es aquel que tenga la puntuación máxima total. Si dos o más caminos alcanzan la misma puntuación, uno es elegido arbitrariamente para representar el mejor alineamiento. La ruta también puede moverse horizontalmente o verticalmente en un punto determinado, que correspondería a la introducción de un hueco, inserción o deleción en una de los dos secuencias. En el siguiente ejemplo, se muestra un alineamiento de dos secuencias utilizando la programación dinámica. Se utiliza un sistema de puntuación básico (matriz identidad) siendo 1 para iguales, 0 para distintos y -1 para penalizaciones (ver Figura 5).

	A	T	T	G	C
A	1	0	0	0	0
G					
G					
C					

	A	T	T	G	C
A	1	0	0	0	0
G	0	1			
G					
C					



Alineamiento final:

ATTGC

A-GGC

Figura 5. Ejemplo de alineamiento de dos secuencias por pares utilizando la programación dinámica.

Penalización por hueco

Realizar un alineamiento óptimo entre secuencias a veces conlleva aplicar huecos que representan inserciones o deleciones. Debido a que en el proceso natural de la evolución la inserción y la deleción son poco frecuentes en comparación con las sustituciones, introducir huecos debería de hacerlo computacionalmente más complicado, reflejando rarezas en los eventos de inserción y supresión en la evolución. Sin embargo, asignar valores de penalización puede ser más o menos arbitrario porque no hay ninguna teoría que determine con precisión el costo de introducir inserciones o deleciones.

Si los valores de penalización son demasiado bajos, los huecos pueden ser muy numerosos para permitir incluso secuencias no relacionadas con emparejamientos con alta puntuación de similitud. Si los valores de penalización son fijados demasiado altos, los huecos pueden llegar a ser muy complicados que aparezcan, sin que se pueda alcanzar un alineamiento razonable, lo que también es poco realista. A través de estudios empíricos para proteínas globulares, ha sido desarrollado un conjunto de valores de penalización que parecen satisfacer los propósitos de la mayoría de alineamientos. Normalmente, suelen aparecer implementados como valores por defecto en la mayoría de programas de alineamiento.

Método heurístico de palabra corta

La búsqueda en una base de datos de gran tamaño utilizando los métodos de programación dinámicos, tales como el algoritmo de Smith-Waterman, es demasiado lento y poco práctico cuando los recursos computacionales son limitados (aunque sea preciso y fiable). Una estimación realizada hace casi una década ha demostrado que consultar una base de datos de 300.000 secuencias utilizando una secuencia problema de 100 residuos, tomó 2-3 horas en completar con un sistema computacional regular. Por lo tanto, la velocidad de búsqueda se convirtió en un tema importante. Para acelerar la comparación, los métodos heurísticos tienen que ser utilizados. Los algoritmos heurísticos realizan búsquedas más rápidas porque examinan sólo una fracción de los posibles alineamientos examinados en la programación dinámica regular.

En la actualidad, hay dos principales algoritmos heurísticos para la realización de búsquedas de bases de datos: BLAST y FASTA. Estos métodos no están garantizados para encontrar el mejor alineamiento u homólogos verdaderos; pero son de 50 a 100 veces más rápidos que la programación dinámica. El aumento de la velocidad de cálculo viene con un gasto moderado de sensibilidad y especificidad de la búsqueda, que es fácilmente tolerado por los trabajadores en biología molecular. Ambos programas pueden proporcionar una indicación razonablemente buena de similitud de secuencias mediante la identificación de segmentos de secuencias similares.

Tanto BLAST y FASTA usan un método heurístico de palabra corta para el alineamiento rápido de secuencias por pares. Éste, es el tercer método de alineamiento de secuencias por pares. Funciona mediante la búsqueda de tramos cortos de letras idénticas o casi idénticas en dos secuencias. Estas cadenas cortas de caracteres se llaman 'palabras cortas', que son similares a las ventanas que se utilizan en el método de matriz por puntos. El supuesto básico, es que dos secuencias relacionadas deben tener al menos una palabra en común; identificando primero que la palabra coincide, se puede obtener un alineamiento más largo mediante la extensión de las regiones de similitud de las palabras cortas. Una vez que son encontradas las regiones de alta similitud de secuencia, las regiones con alta puntuación adyacentes pueden unirse en un alineamiento completo.

El servidor web BLAST [25] ha sido diseñado de forma que simplifica las tareas de los programas seleccionados. Los programas están organizados basándose en el tipo de secuencias de consulta, secuencias de proteínas, secuencias de nucleótidos, o secuencia de nucleótidos a traducir.

El programa FASTA basado en la web y ofrecido por el Instituto Europeo de Bioinformática [26] permite el uso, ya sea de ADN o de secuencias de proteínas, como consultas para búsquedas en una base de datos tanto de nucleótidos como de proteínas.

5.6. Matrices de puntuación

En la programación dinámica de los algoritmos que se van a presentar, el procedimiento para alinear secuencias utiliza un sistema de puntuación que está formado por un conjunto de valores que cuantifican la probabilidad de que un residuo sea sustituido por otro en el alineamiento. A este sistema de puntuación se le conoce como *matriz de sustitución* y deriva del análisis estadístico de la sustitución de conjuntos fiables de secuencias altamente relacionadas de residuos de datos.

Las matrices de puntuación para secuencias de nucleótidos son relativamente simples. Un valor positivo o una alta puntuación es dado por un buen emparejamiento, mientras que un valor bajo o negativo es dado por un mal emparejamiento. La asignación está basada en la suposición de que las frecuencias de mutación son iguales para todas las bases.

Las matrices de puntuación para aminoácidos son más complicadas porque la puntuación tiene que reflejar las propiedades físico-químicas de los residuos aminoácidos; como también la probabilidad de determinados residuos de ser sustituidos entre secuencias homólogas verdaderas. Ciertos aminoácidos con propiedades físico-químicas similares pueden ser más fácilmente sustituidos que otros sin características similares. La sustitución entre residuos similares tiende a preservar las características funcionales y estructurales esenciales. Sin embargo, las sustituciones entre residuos con diferentes propiedades físico-químicas son más propensos a causar alteraciones en la estructura y en la función. Este tipo de sustitución disruptiva es menos probable que se seleccione en la evolución, ya que provee proteínas no funcionales.

5.6.1. Matrices de puntuación para aminoácidos

Las matrices de sustitución de aminoácidos, que son matrices de 20 × 20, se han ideado para reflejar la probabilidad en la sustitución de residuos. Hay esencialmente dos tipos de matrices de sustitución para los aminoácidos. Un tipo se basa en la intercambiabilidad del código genético o de las propiedades de los aminoácidos, mientras que el otro deriva de los estudios empíricos sobre sustituciones de aminoácidos. Aunque las dos diferentes aproximaciones coinciden en cierta medida, el primer enfoque que está basado en el código genético o en las propiedades físico-químicas de los aminoácidos, ha demostrado ser menos exacto que el segundo enfoque, que está basado en estudios actuales sobre la sustitución de aminoácidos entre proteínas relacionadas. Por consiguiente, la aproximación empírica ha ganado mayor popularidad en las aplicaciones de alineamiento de secuencias, siendo el foco de nuestra próxima discusión.

Las matrices empíricas, que incluyen las matrices PAM y BLOSUM, derivan de los alineamientos actuales entre secuencias altamente similares. Mediante el análisis de las probabilidades de sustitución de aminoácidos en estos alineamientos, un sistema de puntuación puede ser desarrollado dando una alta puntuación en la sustitución más probable y una puntuación baja en las sustituciones excepcionales.

Para una matriz de sustitución dada, una puntuación positiva significa que la frecuencia de sustitución encontrada en un conjunto de datos de secuencias homólogas, es mayor que la que se ha producido por casualidad probabilística. Todo ello representa sustituciones de residuos muy similares o idénticos. Una puntuación cero significa que la frecuencia de sustitución de aminoácidos encontrada en un conjunto de secuencias homólogas es igual a lo esperado por probabilidad. En este caso, la relación entre los aminoácidos es débilmente similar en el mejor de los casos en términos de propiedades físico-químicas. Una puntuación negativa significa que la frecuencia de sustitución de aminoácidos encontrada en la secuencia homóloga de un conjunto de datos, es menor que el que se ha producido por casualidad probabilística. Esto normalmente se produce en las sustituciones entre residuos diferentes.

5.6.1.1. Matrices PAM

Las matrices PAM "Point Accepted Mutation" (mutación puntual aceptada) son matrices de sustitución de aminoácidos que describe la probabilidad de que un aminoácido sea sustituido por otro. Es construido calculando en primer lugar el número de sustituciones observadas en un conjunto de secuencias de datos con 1% de mutaciones en aminoácidos y posteriormente extrapolando el número de sustituciones a conjuntos de secuencias de datos más divergentes a través de la duplicación de la matriz. A continuación, se muestra en la Tabla 14 la matriz de sustitución de aminoácidos PAM250, donde los residuos están agrupados según las similitudes físico-químicas.

Tabla 14. Matriz de sustitución de aminoácidos PAM250. Los residuos están agrupados en base a las similitudes físico-químicas.

C	12																			
S	0	2																		
T	-2	1	3																	
P	-3	1	0	6																
A	-2	1	1	1	2															
G	-3	1	0	-1	1	5														
N	-4	1	0	-1	0	0	2													
D	-5	0	0	-1	0	1	2	4												
E	-5	0	0	-1	0	0	1	3	4											
Q	-5	-1	-1	0	0	-1	1	2	2	4										
H	-3	-1	-1	0	-1	-2	2	1	1	3	6									
R	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6								
K	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5							
M	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6						
I	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5					
L	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-2	4	2	6				
V	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4			
F	-4	-3	-3	-5	-4	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9		
Y	0	-3	-3	-5	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	7	10	
W	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17
	C	**S**	**T**	**P**	**A**	**G**	**N**	**D**	**E**	**Q**	**H**	**R**	**K**	**M**	**I**	**L**	**V**	**F**	**Y**	**W**

La unidad PAM está teóricamente relacionada con el tiempo evolutivo, cuya unidad corresponde a 10 millones de años de cambios evolutivos. Por lo tanto, cuanto mayor es la numeración PAM , más divergencia reflejan las secuencias de aminoácidos (ver Tabla 15).

Tabla 15. Los números PAM correspondientes a la observación del ratio de las mutaciones de los aminoácidos.

Número PAM	Tasa de Mutación (%)	Tasa de Identidad (%)
0	0	100
1	1	99
30	25	75
80	50	50
110	40	60
200	75	25
250	80	20

5.6.1.2. Matrices BLOSUM

La matriz de sustitución de aminoácidos es construida a partir de las frecuencias de sustitución observadas en los bloques de alineamiento de secuencias sin huecos de proteínas estrechamente relacionadas. La numeración de las matrices BLOSUM corresponde al porcentaje de identidad en los bloques de secuencias de proteínas. A continuación, en la Tabla 16 se muestra la matriz de sustitución de aminoácidos BLOSUM62.

Tabla 16. Matriz de sustitución de aminoácidos BLOSUM62.

C	9																			
S	-1	4																		
T	-1	1	5																	
P	-3	-1	-1	7																
A	0	1	1	1	4															
G	-3	0	0	-1	1	6														
N	-3	1	0	-1	0	0	6													
D	-3	0	-1	-1	-2	-1	1	6												
E	-4	0	-1	-1	-1	-2	0	2	5											
Q	-3	0	-1	-1	-1	-2	0	0	2	5										
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8									

R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5								
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5							
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5						
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4					
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4				
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4			
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6		
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-3	-2	-1	-1	-1	-1	3	7	
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11
	C	**S**	**T**	**P**	**A**	**G**	**N**	**D**	**E**	**Q**	**H**	**R**	**K**	**M**	**I**	**L**	**V**	**F**	**Y**	**W**

5.6.1.3. Comparación entre PAM y BLOSUM

Hay una serie de diferencias entre PAM y BLOSUM. La diferencia principal es que las matrices PAM, excepto PAM1, derivan de un modelo evolutivo mientras que las matrices BLOSUM consisten únicamente en observaciones directas. Por lo tanto, las matrices BLOSUM pueden tener un menor significado evolutivo que las matrices PAM. Esta es la razón por la que las matrices PAM son las más utilizadas para la reconstrucción de árboles filogenéticos. Sin embargo, debido al procedimiento de extrapolación matemática utilizada, los valores PAM pueden ser menos realistas para secuencias divergentes. Las matrices BLOSUM derivan por completo de las secuencias de alineamiento local de bloques de secuencias conservadas; mientras que la matriz PAM1, se basa en el alineamiento global de secuencias de longitud completa compuestas por regiones conservadas y variables. Esta es la razón por las que las matrices BLOSUM pueden ser más ventajosas en búsquedas en bases de datos y en la localización de dominios conservados en las proteínas.

Varias pruebas empíricas han demostrado que las matrices BLOSUM superan a las matrices PAM en términos de precisión en el alineamiento local. Esto podría ser en gran medida debido a que las matrices BLOSUM derivan de un conjunto de datos más representativo y mucho más grande que la que se utiliza para derivar las matrices PAM. Esto se traduce en mayor

fiabilidad para los valores de las matrices BLOSUM. Para compensar las deficiencias en el sistema PAM, se han diseñado nuevas matrices usando el mismo enfoque pero basadas en conjuntos de datos mucho más grandes. Estas incluyen a las matrices de Gonnet y a las matrices de Jones-Taylor-Thornton, que se ha demostrado que tienen un rendimiento equivalente a BLOSUM en alineamientos regulares, pero que son particularmente robustas en la construcción de árboles filogenéticos.

5.7. Importancia de la estadística en el alineamiento de secuencias

Cuando se presenta un alineamiento de secuencia que muestra cierto grado de similitud, a menudo es importante preguntarse si el alineamiento de secuencia observado ha ocurrido al azar o si el alineamiento es un hecho estadísticamente sólido. El alineamiento de secuencia estadísticamente significativo será capaz de proporcionar pruebas de homología entre las secuencias implicadas.

La solución de este problema requiere un análisis estadístico de las puntuaciones del alineamiento de dos secuencias no relacionadas con la misma longitud. Mediante el cálculo de puntuaciones del alineamiento de un gran número de pares de secuencias no relacionadas, se puede derivar un modelo de distribución de puntuaciones de secuencia aleatorias. A partir de la distribución, se puede realizar una prueba estadística basándose en el número de desviaciones estándar de la puntuación media. Muchos estudios han demostrado que la distribución de las puntuaciones de similitud asume una forma peculiar que se asemeja a una distribución normal altamente sesgada con una larga cola en un lado.

La distribución coincide con la "distribución de valores extremos Gumbel" para lo que hay una expresión matemática disponible. Esto significa que dado un valor de similitud de secuencia, mediante el uso de la fórmula matemática para la distribución extrema, la significación estadística se puede estimar con precisión (ver Figura 6).

La prueba estadística para la relación de dos secuencias puede realizarse utilizando el siguiente procedimiento: Se obtiene primero un alineamiento óptimo entre dos secuencias dadas , para a continuación, generar secuencias no relacionadas de la misma longitud a través de un proceso de aleatorización en el que una de las dos secuencias se pone en orden aleatorio. Una nueva puntuación de alineamiento se calcula entonces para el par de secuencias barajadas, obteniendo más de estos resultados de manera similar a través de repetidas mezclas. El grupo de puntuaciones del alineamiento de secuencias mezcladas se utiliza para generar parámetros para la distribución extrema.

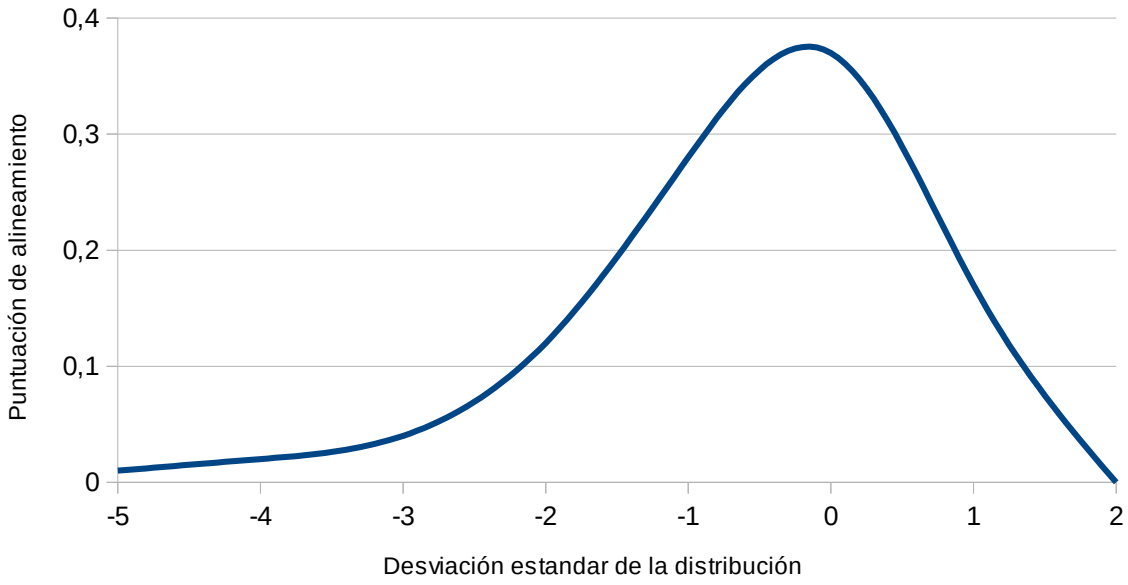


Figura 6. Distribución de valores extremos Gumbel para puntuaciones de alineamiento.

La puntuación del alineamiento original es a continuación comparada con la distribución de alineamientos al azar para determinar si la puntuación va más allá del azar probabilístico. Si la puntuación se encuentra en el margen extremo de la distribución, lo que significa que el alineamiento entre las dos secuencias es poco probable debido a la casualidad del azar y, por lo tanto, se considera significativa. El valor *P* es dado para indicar la probabilidad de que el alineamiento original se deba al azar probabilístico.

El valor *P* resultante de la prueba, proporciona un indicador mucho más fiable de las posibles relaciones homólogas que usando los valores del porcentaje de identidad. Por tanto, es importante saber cómo interpretar los valores *P*. Se ha demostrado que si un valor *P* es menor que 10^{-100}, indica una coincidencia exacta entre las dos secuencias. Si el valor *P* está en el rango de 10^{-50} a 10^{-100}, se considera que es un partido casi idéntico. Un valor *P* en el intervalo de 10^{-5} a 10^{-50} se interpreta como secuencias que tienen una clara homología. Un valor *P* en el rango de 10^{-1} a 10^{-5} indica posibles homólogos distantes. Si *P* es mayor que 10^{-1} las dos secuencias pueden estar relacionadas al azar. Sin embargo, el inconveniente es que a veces las secuencias de proteínas verdaderamente relacionadas, pueden carecer de significación estadística a nivel de secuencia debido a las tasas de divergencia rápida. Sus relaciones evolutivas, sin embargo, pueden ser reveladas a nivel estructural tridimensional.

Estos datos se obtuvieron a partir de secuencias de alineamiento locales sin huecos. No se sabe si la distribución Gumbel se aplica igualmente bien a los alineamientos con huecos. Sin embargo, para todos los propósitos prácticos, es razonable asumir que las puntuaciones para alineamientos con huecos se ajustan esencialmente a la misma distribución [27].

6. Agentes inteligentes

El uso de agentes inteligentes en el proyecto permite distribuir el alineamiento de pares de secuencias generadas por el usuario para alinearlas, además de escalar, de forma transparente al usuario, los recursos de cómputo disponibles. Ello soporta un sistema de balanceo de carga que incrementa el rendimiento del sistema y le hace flexible a cambios en la configuración de los dispositivos de hardware. En este capítulo se explican los conceptos y fundamentos básicos sobre su desarrollo, además de enumerar y comentar las diferentes herramientas utilizadas tanto para el desarrollo como para su posterior ejecución. El resultado de este trabajo es la creación del sistema "MASBioseq", que incorpora el entorno R junto con su intérprete, los scripts para el alineamiento de secuencias, las librerías dinámicas y todas las librerías R y JAVA necesarias para su correcto funcionamiento.

El agente inteligente se define como una entidad individual y autónoma de software que mediante su propio conocimiento realiza un conjunto de procesos y operaciones con el fin de satisfacer las necesidades tanto de un usuario como de una aplicación, ya sea por iniciativa propia o por requisito de alguno de éstos (Figura 7) [28].

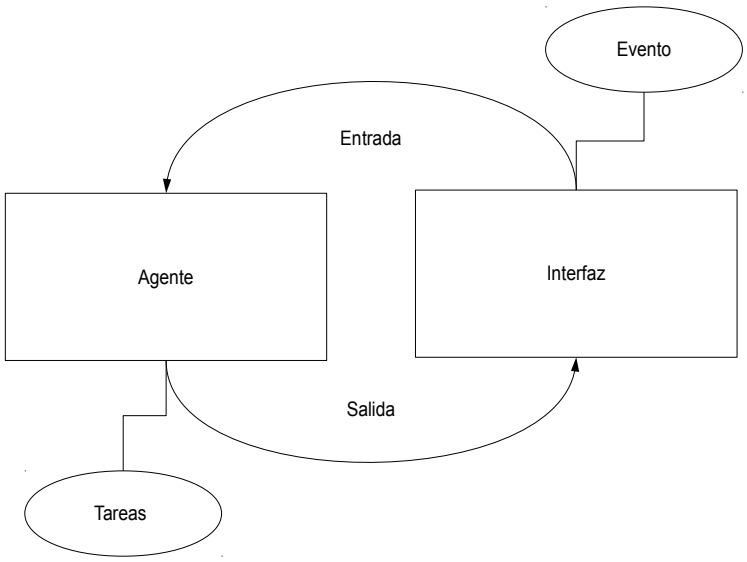


Figura 7. Interacción de un agente de software con su entorno.

Para satisfacer las necesidades del usuario y de la aplicación, los agentes deben tener la capacidad de recoger los eventos y operar con la información de forma autónoma. Ello implica ser capaces de iniciarse para poder realizar las acciones necesarias con la finalidad de solventar el problema encomendado; y cooperar mediante un lenguaje común con el resto de los agentes del sistema, si fuera necesario para la consecución del objetivo por el que trabajan. De esta forma, se consigue un sistema multi-agente que posibilita la distribución de tareas y la cooperación mediante el paso de información.

6.1. Java Agent Development Framework (JADE)

Para el desarrollo de un sistema multi-agente, se ha utilizado JADE [29]. Se trata de un framework que cumple con las especificaciones de la "Foundation for Intelligent Physical Agents" (FIPA) [30] desarrollado en JAVA y distribuido como software libre.

6.1.1. Plataforma

El modelo de plataforma de agentes está definido por defecto por FIPA y representado por diferentes entidades, sistemas y servicios representados en el gráfico a continuación (ver Figura 8).

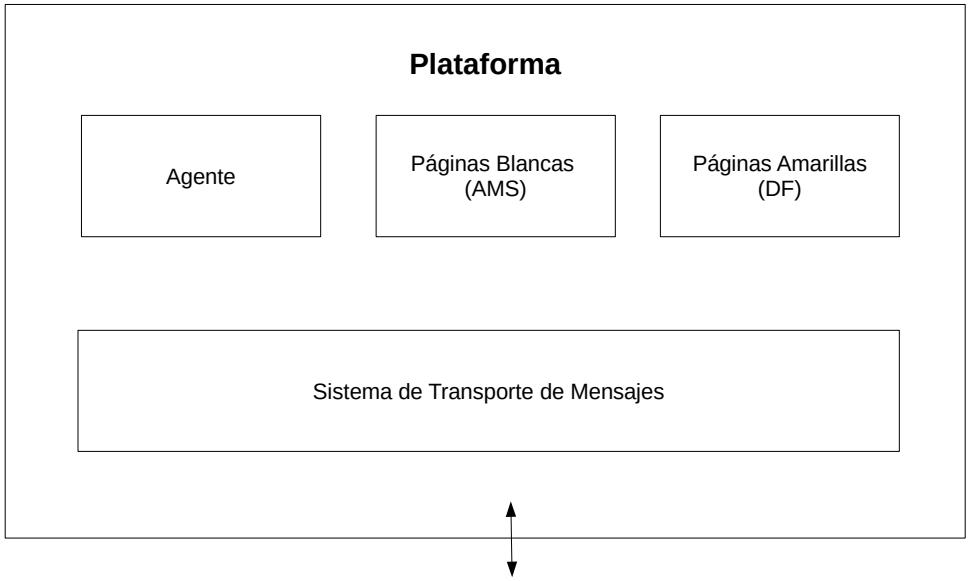


Figura 8. Arquitectura estándar de una plataforma de agentes.

El sistema de administración de agentes (AMS), es un agente que se encarga de controlar y supervisar el acceso y la utilización de la plataforma. Para ello, dispone de un servicio de Páginas Blancas donde se encuentran registrados e identificados los agentes y su respectivo estado dentro del sistema. El facilitador de directorios (DF), conocido como servicio de Páginas Amarillas, es el agente encargado de clasificar a los agentes (una vez registrados en el DF) por los servicios o "tareas" que necesitan o que ofrecen a otros agentes del sistema. Para finalizar, el sistema de transporte de mensajes gestiona el intercambio de mensajes de la plataforma, tanto a nivel local como a nivel distribuido (externo).

Una plataforma está asociada normalmente a cada uno de los ordenadores del sistema distribuido. La plataforma incluye a su vez una serie de contenedores, cada uno de los cuales contiene una serie de agentes. Hay un contenedor principal que es el que mantiene los servicios de Páginas Amarillas y Páginas Blancas. A estos se conectarán el resto de contenedores y

agentes para hacer posible la interoperatividad y el funcionamiento a nivel distribuido del sistema y del entorno de ejecución (Figura 9).

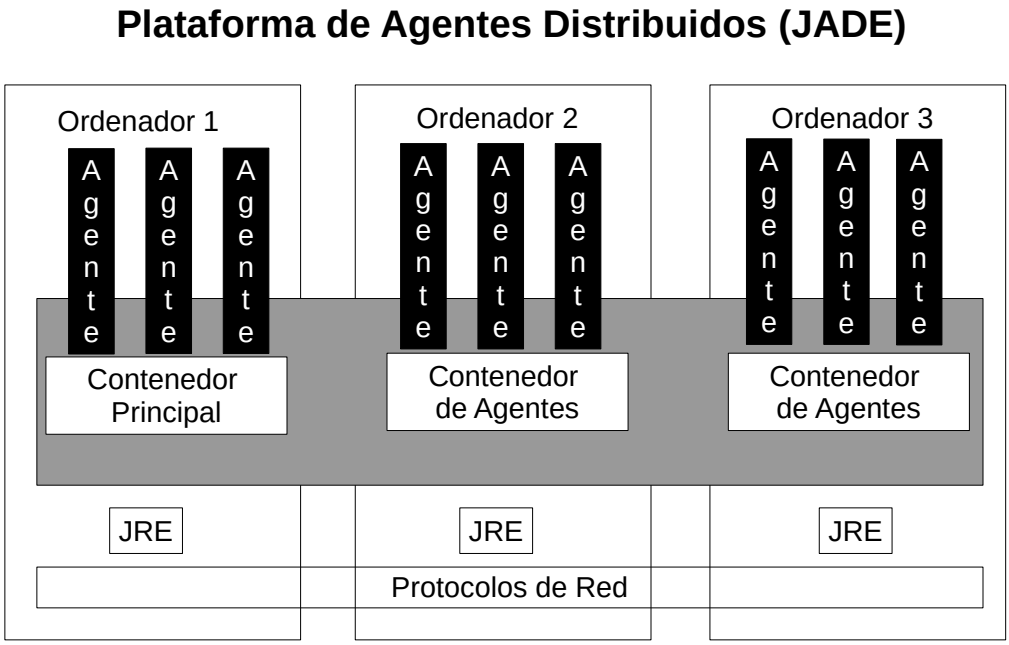


Figura 9. Plataforma distribuida sobre varios contenedores.

Páginas Amarillas (DF)

El servicio DF (*Directory Facilitator*), como se ha comentado anteriormente, permite a los agentes registrarse y publicar los servicios que proporcionan para que otros agentes puedan acceder a ellos (ver Figura 10).

Los agentes interaccionan con el DF intercambiando mensajes. Para ello, JADE dispone de una serie de métodos de la clase *DFService* desde donde el agente puede publicar o solicitar un servicio. Para ello, tiene que proporcionar mediante el objeto de la clase *DFAgentDescription* el identificador (AID) del agente. Respecto a los servicios que ofrece, debe proporcionárselos al objeto de la clase *ServiceDescription*. Por cada servicio, es obligatorio asignar un nombre y un tipo.

Para realizar las acciones anteriores, se utilizan los siguientes métodos de la clase *DFService*:

- **register()**: Registra los servicios de un agente. Este método se utiliza en el proyecto para registrar los agentes que estén disponibles para alinear secuencias.
- **deregister()**: elimina del registro los servicios del agente. En el proyecto, lo utilizan los agentes que se encuentran realizando alineamientos para marcar que están ocupados.

Para la definición de los servicios, se han utilizado los siguientes métodos de la clase *ServiceDescription*:

- **setName()**: Añade un nombre obligatorio al servicio.
- **setType()**: Añade un tipo obligatorio al servicio.

Para modificar la descripción del agente, se han utilizado los siguientes métodos de la clase *DFAgentDescription*:

- **setName()**: Añade el AID del agente.
- **addServices()**: Añade el agente y su servicio correspondiente al registro.

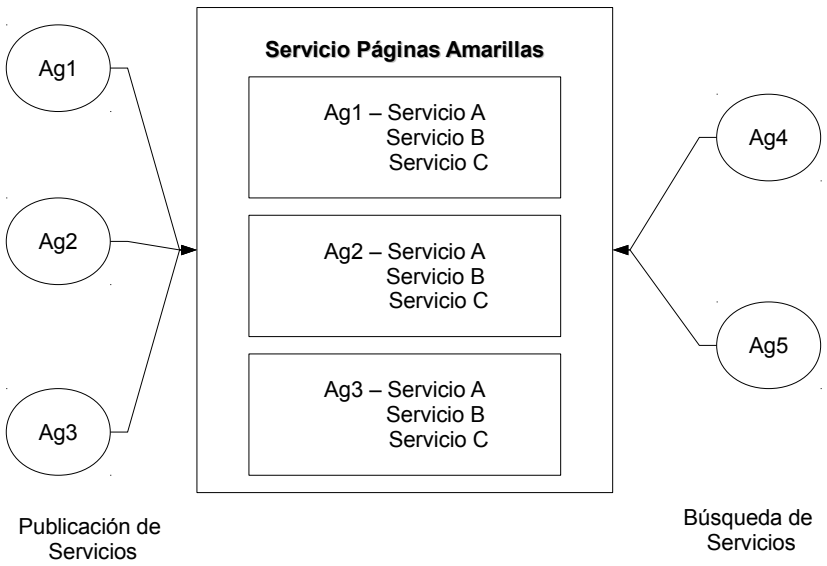


Figura 10. Servicio de Páginas Amarillas.

Un agente que solicita servicios, solamente necesita realizar una búsqueda al *DFAgentDescription* mediante el método `search()` de la clase *DFService*. La búsqueda devuelve una lista con todas las descripciones en base a una plantilla *DFAgentDescription* dada.

Páginas Blancas (AMS)

El servicio de páginas blancas (AMS) garantiza que cada agente disponga de un nombre único. Además, proporciona y mantiene el directorio de los identificadores de agentes (AID) y su estado en el ciclo de vida. Cada agente tiene la obligación de registrarse con el AMS si desea obtener un AID válido. Esta operación en JADE la realizan los agentes de forma automática al ejecutarse el método `setup()`, por lo que el usuario no tiene que preocuparse de nada. Para acceder a los servicios AMS es necesario importar la clase *AMSService* [31].

6.1.2. Clase Agente

Para programar y representar un agente en JADE se debe definir primeramente una clase que herede de *Agent*. Posteriormente, se implementarán los comportamientos que realizará.

Un agente en JADE tiene un nombre único que lo define en el entorno donde está ejecutándose. Además, es implementado como un único hilo de ejecución. El agente, dispone de un método de inicialización `setup()` que es invocado al comienzo de su ejecución y que es utilizado para inicializar el agente junto con los comportamientos asignados. Es necesario implementar una clase interna para cada uno de los comportamientos. Estos comportamientos se utilizan, entre otras tareas, para el envío y su posterior recepción de mensajes entre los agentes del sistema.

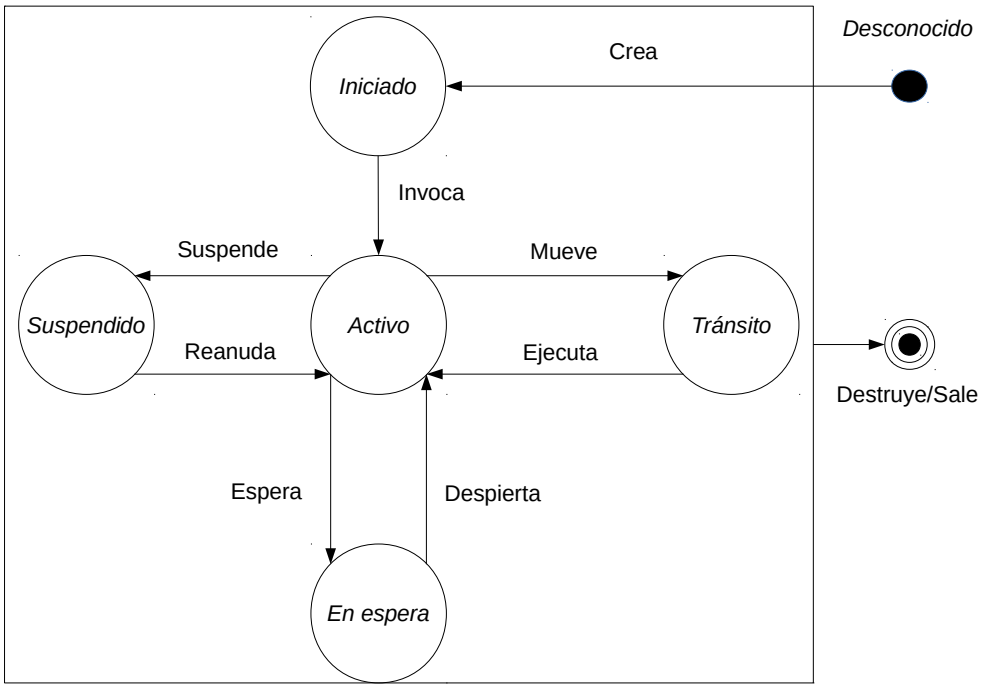


Figura 11. Ciclo de vida de un agente.

Un agente puede estar en los siguientes estados (Figura 11):

- **Iniciado**: El agente ha sido creado pero sin registrarse en el AMS (Agent Management System). Por lo tanto, no dispone ni de nombre, ni de dirección, ni tampoco puede interactuar con otros agentes.
- **Activo**: El agente ha sido registrado en el AMS. Dispone de nombre, dirección y con acceso a cualquier opción de JADE.
- **Suspendido**: El agente se encuentra parado, puesto que su hilo de ejecución ha sido detenido, y por lo tanto, no puede ejecutar ningún comportamiento.
- **En espera**: El agente se encuentra bloqueado y en espera. Su hilo de ejecución se encuentra dormido y se despertará al cumplir ciertas condiciones.
- **Desconocido**: El agente es eliminado del sistema y del registro AMS, junto con su correspondiente hilo de ejecución.

Una vez que el agente es creado, automáticamente se llama al constructor del agente y se crea un identificador (AID). Posteriormente se registra en el AMS y se ejecuta el método `setup()`. El método `setup()`, además de albergar los comportamientos que ejecutará el agente, tiene la capacidad de modificar el registro del AMS o de registrarse en el DF (*Directory Facilitator*) entre otras muchas acciones.

Comunicación

La clase *Agent* dispone de una serie de métodos para el envío y recepción de mensajes asíncronos donde los agentes podrán intercambiar información. En el proyecto, se han introducido e intercambiado DataStores **(**almacén de objetos) entre los agentes mediante los mensajes. Los métodos utilizados para el envío y recepción de mensajes por parte de los agentes son los siguientes:

- **send():** El agente emisor envía un mensaje al agente o agentes receptores especificados dentro del mensaje.
- **receive():** El agente recibe el mensaje enviado por el agente emisor. Es posible filtrar los mensajes a recibir analizando parámetros propios que contienen los mensajes.

Interfaz GUI

Las interfaces gráficas de usuario (GUI) no pueden ser tratadas como agentes en principio al carecer de autonomía. Los agentes trabajan con ellas mediante la clase *GuiEvent* de JADE. Ésta representa los eventos que dispara el usuario en la interfaz usando sus componentes Mediante el objeto GuiEvent, se es capaz de operar dentro de los eventos de la interfaz. Se pueden añadir al evento los parámetros que se consideren oportunos mediante el método `addParameter()`, para después enviarlo al agente que procesará esa información mediante el método `postGuiEvent(GuiEvent)`. Para hacer esto posible, será necesario definir previamente como argumento de la interfaz el agente que procesará el objeto GuiEvent enviado.

El agente encargado de recibir esa información, tendrá que instanciar dentro de su método `setup()` un objeto que represente la interfaz GUI donde se pase el agente como parámetro. En el método sobrescrito `onGuiEvent()`, el agente recibirá el objeto GuiEvent enviado desde la interfaz. Es necesario saber que la información almacenada en el evento, se encuentra en el orden en el que hayan sido introducidos los parámetros; y que se accede mediante el método `getParameter(int)`.

6.1.3. Lenguaje de Comunicación entre Agentes (ACL)

Para posibilitar el intercambio de mensajes entre agentes, será necesario utilizar la clase *ACLMessage* para crear un objeto ACLMessage al que configurar los atributos que se consideren necesarios.

A continuación, se listan una serie de atributos asociados al mensaje ACL utilizados en nuestro proyecto:

- **performative:** indica el tipo de acto del habla a realizar en la comunicación (acción que realiza el mensaje). Es el único campo obligatorio y puede tomar uno de los siguientes valores (entre otros muchos):
    - **inform**: informar a un receptor de que una proposición es cierta.
    - **request**: solicitar a un receptor que realice alguna acción.
- **sender:** AID del emisor.
- **receiver:** lista de AID's de los receptores**.**
- **content:** contenido del mensaje.
- **contentObject:** objeto que contiene el mensaje.

Como ya se ha comentado anteriormente, el intercambio de mensajes ACL es asíncrono, y cada agente dispone de su propia cola de mensajes entrantes. Además de los atributos, la clase ACLMessage dispone de una serie de métodos. Los utilizados en el proyecto son los siguientes:

- **setPerformative**(): Indica la performativa del mensaje.
- **addReceiver**(): Añade el AID de un agente receptor a la lista de receptores.
- **getAllReceiver**(): Devuelve un iterador con la lista de todos los agentes receptores.
- **SetContentObject():** Indica el objeto a incluir en el mensaje.
- **GetContentObject():** Devuelve el objeto que contiene el mensaje.

Debido a que todos los mensajes deben contener al menos la performativa, se recomienda indicarlo en el constructor de la instancia ACLMessage. Una vez que se haya creado el objeto ACLMessage, se usarán los métodos disponibles para rellenar los campos que se consideren necesarios. Finalmente, habrá que llamar al método `send()` de la clase *Agent* que recibirá como parámetro un ACLMessage que enviará a los destinatarios que le hayamos indicado. Para la recepción de mensajes, se utiliza el método `receive()` de la clase *Agent*, que obtiene el primer mensaje de la cola de mensajes que cumpla el criterio establecido (devolviendo `null` si está vacía o no hay ningún mensaje apropiado).

Para filtrar los mensajes que un agente desea recibir, con el objetivo de lograr una correcta distribución y envío de mensajes entre agentes, se utiliza la clase *MessageTemplate*. Esta clase define filtros para cada atributo del mensaje ACLMessage y es utilizado como parámetro en el método `receive()`. Las opciones de filtrado utilizadas en nuestro caso, han sido para comprobar el atributo *performative,* ya que a lo largo de la ejecución los agentes reciben mensajes continuamente desde diferentes emisores. De esta manera, tenemos la posibilidad de recoger un mensaje en concreto y realizar las acciones que el agente considere oportunas. Se ha utilizado el siguiente método de la clase *MessageTemplate*:

- **MatchPerformative** (*performative*) donde *performative* puede ser:
    - ACLMessage.INFORM,
    - ACLMessage.QUERY.

6.1.4. Comportamientos: Las tareas de los agentes.

Un agente tiene la posibilidad de tener ejecutando más de una tarea de forma concurrente. Esas tareas, a las que se denominan comportamientos (o *behaviours*), hacen referencia a una funcionalidad que incorpora el agente donde se especifican las tareas o los servicios que ha de realizar.

Un comportamiento debe definirse a través de la clase Behaviour. Estos comportamientos una vez instanciados, se añadirían a la lista de tareas del agente específico. Para asignar un comportamiento a un agente, bastaría con utilizar el método `addBehaviour(Behaviour)` disponible en la clase *Agent*, que lo incluiría en la cola de comportamientos del agente a ejecutar por su planificador. El planificador los ejecutaría utilizando la política 'round-robin' [32] de una estructura FIFO. Los comportamientos pueden ser añadidos o eliminados en cualquier momento durante la ejecución, ya sea desde el método `setup()` del agente o desde cualquier otro comportamiento.

El planificador del agente ejecuta el método `action()` de cada comportamiento presente en la cola del agente; por lo que se deberá implementar dicho método. Este método define la acción a ejecutar por el comportamiento. Debido a que mientras que se ejecuta este método no puede ser interrumpido por otros comportamientos, se utiliza el método `block()` para bloquearlo hasta que reciba un nuevo mensaje y así liberar ciclos de computación. Este método no influye en los demás comportamientos que esté ejecutando el agente.

Existen una gran variedad de clases de comportamientos abstractos que heredan de la clase *Behaviour*. En el proyecto se ha trabajado con las siguientes clases abstractas:

- **SimpleBehaviour:** Representa comportamientos atómicos simples. De esta clase heredan las siguientes:
    - **OneShotBehaviour:** Comportamiento que se ejecuta una sola vez.
    - **CyclicBehaviour:** Comportamiento cíclico [33].

6.2. Librería RCaller

Con la intención de manipular en JAVA los objetos que el intérprete de R almacena en memoria, hacemos uso de la librería RCaller [34]. Esta librería para JAVA nos devuelve los objetos en formato XML utilizando el paquete Runiversal [35] de R. La función del documento XML no es otra que la de transformar y devolver los objetos como arrays y matrices.

En el proyecto, se utiliza esta librería desde JAVA para cargar objetos en memoria utilizando el intérprete de R, y para ejecutar los diferentes scripts orientados al alineamiento de secuencias. Una vez que se almacenan en memoria los resultados de la ejecución del script, se recuperan en JAVA para que los agentes puedan tratar y procesar esa información.

Esta librería dispone de dos objetos (RCaller y RCode) que junto a una serie de métodos, posibilitarán la interacción de R con JAVA y viceversa. Los métodos asociados al objeto RCaller utilizados, son los siguientes:

- **setRscriptExecutable():** Permitirá mediante una ruta localizar el interprete de R para poder comprender el código R enviado desde JAVA,
- **addRSource():** Carga mediante la ruta definida la localización del script en R a ejecutar.
- **addRCode():** Indica código en R que ejecutará el intérprete desde JAVA.
- **runAndReturnResults():** Devuelve como parámetro el objeto definido por el usuario y que se encuentra en la memoria de R tras la ejecución del script o del código cargado en el objeto RCaller.
- **getParser():** Devuelve el objeto indicado por el usuario cargado tras la ejecución como array o matriz.
- **runOnly():** Ejecuta el script o el código cargado en el objeto RCaller.
- **setRCode()**: Se le pasa como argumento un objeto RCode que leerá como código.

Por otra parte, el objeto RCode dispone de métodos que hemos usado a lo largo de nuestro proyecto:

- **R_Require():** Es utilizado para indicar la librería que ha de ser cargada en R.
- **startPlot():** Indica el inicio del código que se utilizará para la construcción de la gráfica definida por el usuario.
- **endPlot():** Indica el fin del código que se utilizará para la construcción de una gráfica definida por el usuario.

6.3. Arquitectura de la aplicación

En la aplicación "MASBioseq" que se ha desarrollado, se han implementado dos tipos de agentes a los que se les asigna diferentes comportamientos para hacer posible el alineamiento de secuencias. Para ello, el usuario dispone de una interfaz desde donde introduce los distintos parámetros que recogerán y manipularán posteriormente los agentes, y a donde finalmente le enviarán y mostrarán los resultados del alineamiento (Figura 12).

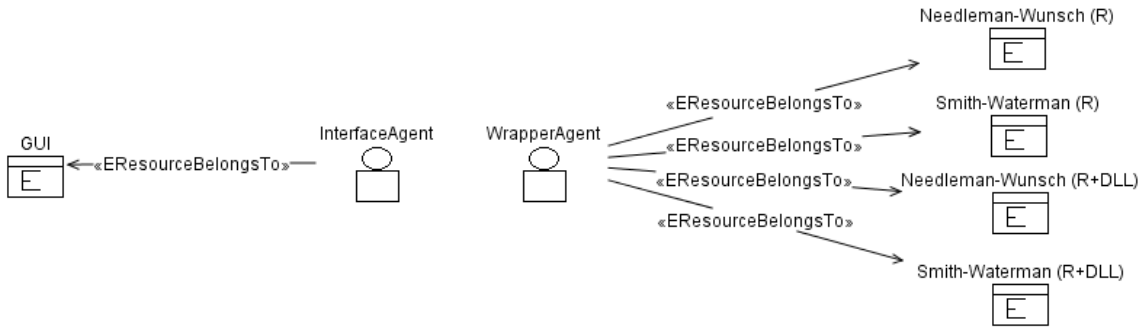


Figura 12. Diagrama de Entorno de "MASBioseq".

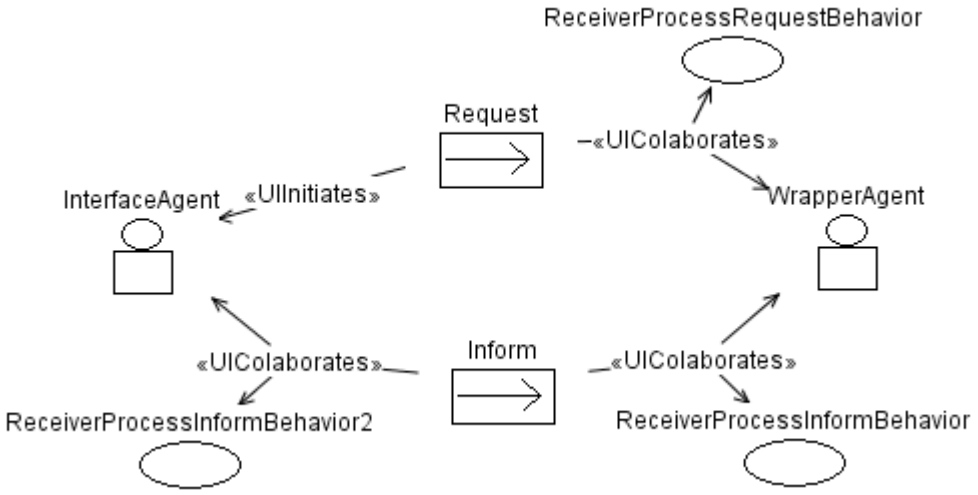


Figura 13. Diagrama de Interacción de "MASBioseq".

Inicialmente, una vez que la aplicación es lanzada, se levantan los siguientes agentes asociados a unas clases específicas. A cada agente se le asignan una serie de comportamientos (ver Figura 13) o tareas con el fin de alcanzar el objetivo común (alinear las secuencias definidas por el usuario a través de la interfaz inicial):

- **GUIAgent:** Primer agente asociado a la clase *InterfaceAgent*. Este agente es el encargado de recoger los parámetros enviados por el usuario, y enviarlos al *WrapperAgent1* o al *WrapperAgent2*. Localiza estos agentes mediante el servicio de páginas amarillas independizando al *GUIAgent* de cuántos y quiénes *WrapperAgent* hay en el sistema, seleccionando el primero disponible. Finalmente, se encarga de recoger los resultados del alineamiento enviados por el agente *WrapperAgent1* o *WrapperAgent2* para mostrarlos en la interfaz final al usuario.

- **WrapperAgent 1 y 2:** Segundo y tercer agente asociados a la clase *WrapperAgent*. Uno de estos dos agentes, recibe los parámetros del agente *GUIAgent*, y crea un hilo (thread) con la ejecución del script. En ese momento, el *WrapperAgent* elegido se deregistra del DF para balancear la carga del sistema. Una vez que el script finaliza, devuelve los resultados al agente *WrapperAgent* y éste, los reenvía al agente *GUIAgent* para su

posterior publicación. Una vez que las tareas del *WrapperAgent* elegido han finalizado, vuelve a registrarse en el DF en espera de recibir nuevos parámetros del *GUIAgent*.

En relación a las herramientas utilizadas para el desarrollo y funcionamiento de la aplicación, Eclipse Kepler [36] ha sido el entorno de desarrollo open source elegido para la implementación en lenguaje JAVA. Las librerías externas (jar) integradas en el IDE, han sido RCaller y JADE. No hay que olvidar que es necesario para la ejecución de cualquier aplicación desarrollada en JAVA, tener instalado el conjunto de utilidades Java Runtime Environment (JRE) [37]. Por otra parte, ha sido necesario modificar los scripts de alineamiento de secuencias iniciales para adaptarlos a nuestras nuevas necesidades. Debido a que la aplicación necesita tener el interprete de R instalado, ha habido que instalar la versión 32 bits de R (la versión de 64 bits es incompatible con las librerías dinámicas de los scripts) junto con el paquete Runiversal para la correcta ejecución de la aplicación.

6.3.1. Ejemplo de ejecución

La configuración de ejecución de JADE (Run/Run Configurations/Java Application), contiene una serie de parámetros utilizando como clase principal `jade.boot` (Figuras 14 y 15), que instanciará los agentes y la interfaz JADE Remote Agent Management GUI (ver Figura 16) desde donde se puede controlar y realizar un seguimiento de los agentes en todo momento durante la ejecución.

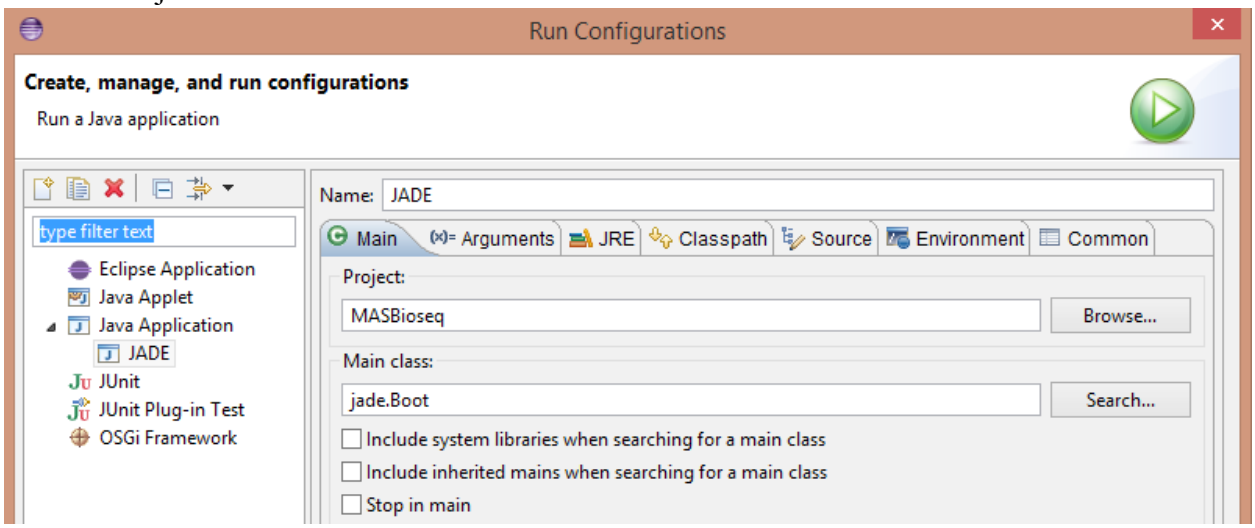


Figura 14. Pestaña Main del Run Configurations donde seleccionamos *jade.Boot*.

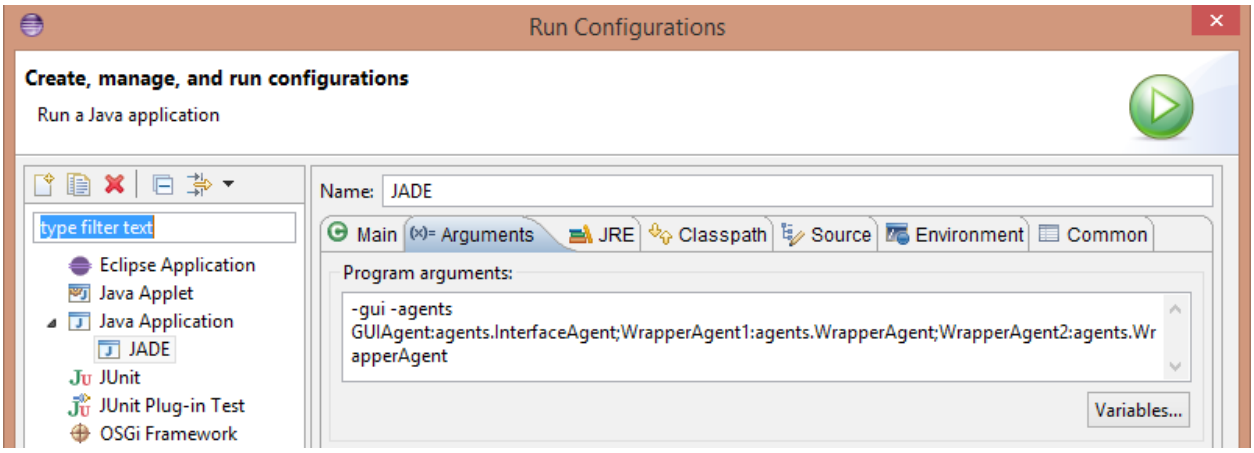


Figura 15. Argumentos para la ejecución de agentes desde clases específicas.

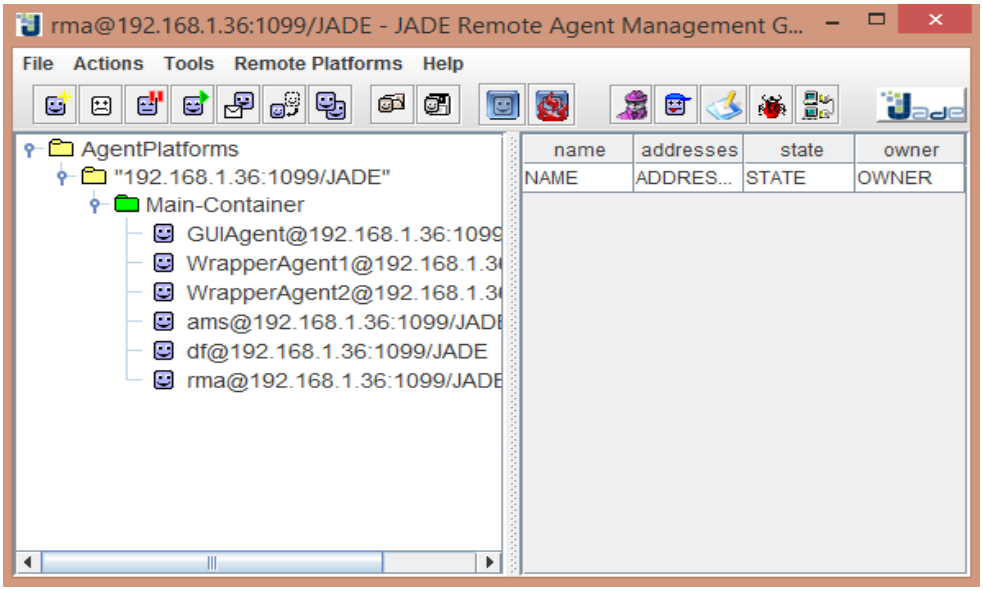


Figura 16. JADE Remote Agent Management GUI.

Una vez comenzada la ejecución, aparecerá en pantalla las siguientes tres interfaces (Figuras 17, 18 y 19).

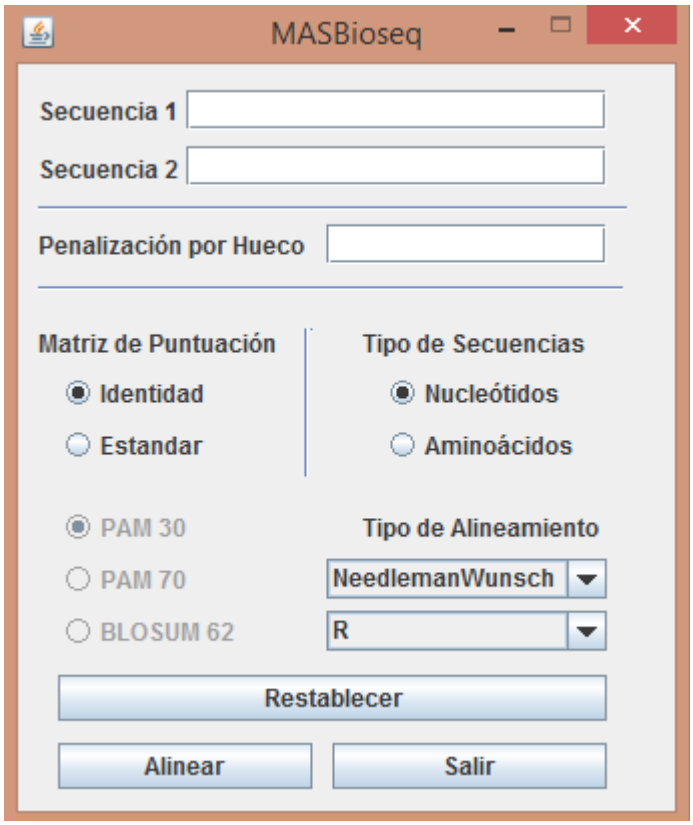


Figura 17. Interfaz inicial para el alineamiento de secuencias.

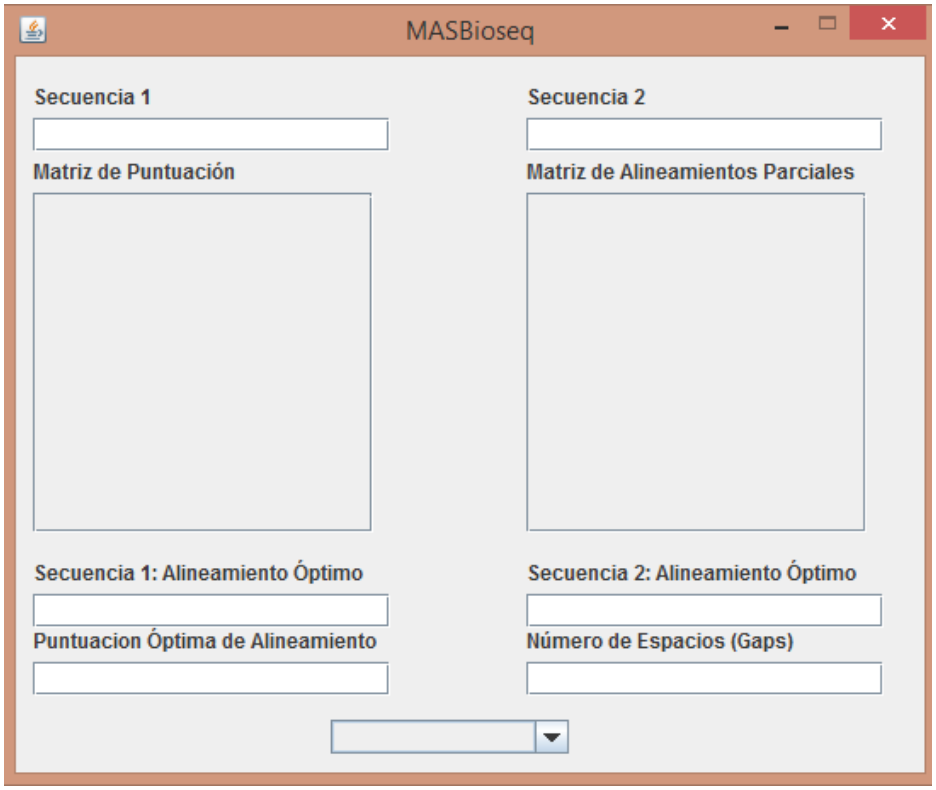


Figura 18. Interfaz resultado para el alineamiento de secuencias.

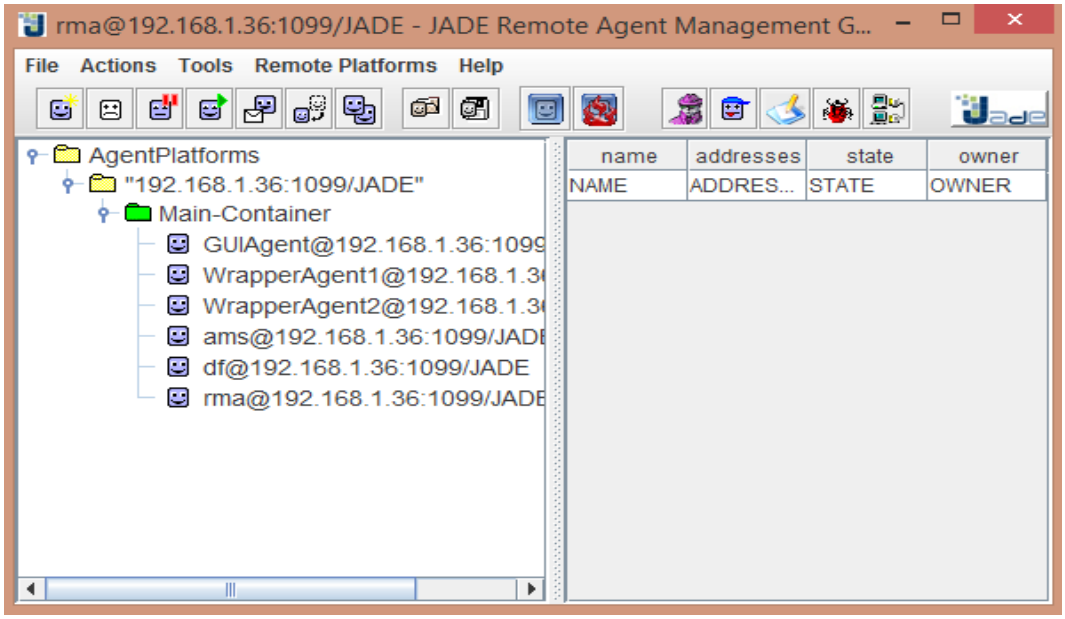


Figura 19. JADE Remote Agent Management GUI.

Interfaz Inicial

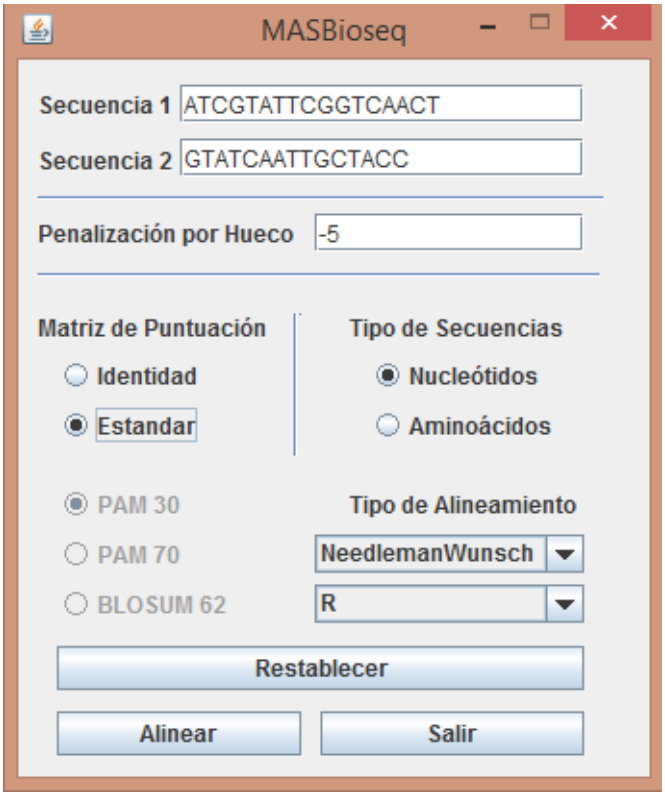


Figura 20. Interfaz Inicial.

Desde la interfaz inicial (ver Figura 20) el usuario puede introducir los parámetros necesarios para comenzar con el proceso de alineamiento de secuencias.

- **Secuencia 1:** Primera secuencia a alinear. Deberán introducirse los caracteres seguidos y sin espacios.
- **Secuencia 2:** Segunda secuencia a alinear. Deberán introducirse los caracteres seguidos y sin espacios.
- **Penalización por Hueco:** Número entero sin decimales que indica la penalización por hueco (gap) que se aplicará a las secuencias.
- **Matriz de Puntuación:** Elección de la matriz de puntuación o de substitución a utilizar durante el alineamiento propuesto (identidad o estándar).
- **Tipo de Secuencias:** Decidir de acuerdo a lo introducido en los campos de las secuencias 1 y 2, el tipo de cadenas que se alinearán (aminoácidos o nucleótidos).
- **Tipo de Alineamiento:** Desplegables para seleccionar el script, a elegir entre el script global (NeedlemanWunsch) o el local (SmithWaterman) en sus implementaciones solamente en R o con librerías dinámicas (DLL).
- **Restablecer:** Limpia y selecciona las opciones por defecto de la interfaz inicial.
- **Alinear:** Comienza a alinear las secuencias introducidas en base a los argumentos introducidos por el usuario.
- **Salir:** Cierra la aplicación.

Interfaz Resultado

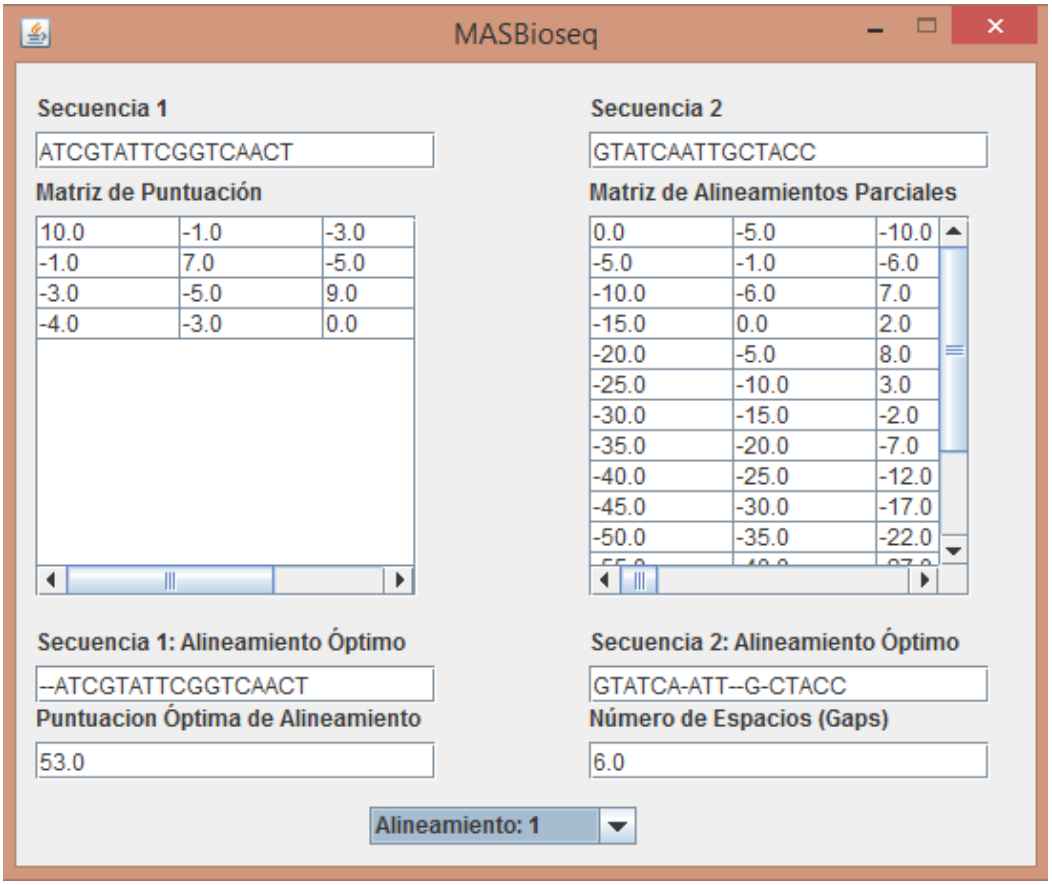


Figura 21. Interfaz Resultado.

Desde la interfaz resultado (Figura 21) el usuario podrá visualizar el resultado final del proceso de alineamiento de las secuencias indicadas, incluyendo la matriz de puntos (ver Figura 21). En el desplegable inferior, se podrán visualizar los resultados de las diferentes secuencias alineadas a medida que los agentes vayan terminando con las tareas encomendadas.

- **Secuencia 1:** Primera secuencia inicial antes de ser alineada.
- **Secuencia 2:** Segunda secuencia inicial antes de ser alineada.
- **Matriz de Puntuación:** Matriz de sustitución seleccionada para el alineamiento.
- **Matriz de Alineamientos Parciales:** Matriz con los cálculos realizados por el script para hallar las secuencias finales.
- **Secuencia 1: Alineamiento Óptimo:** Primera secuencia alineada.

- **Secuencia 2: Alineamiento Óptimo:** Segunda secuencia alineada.
- **Puntuación Óptima de Alineamiento:** Puntuación óptima del alineamiento recogida de la matriz de alineamientos parciales.
- **Número de Espacios (Gaps):** Número de espacios (gaps) utilizados para el alineamiento entre las dos secuencias.

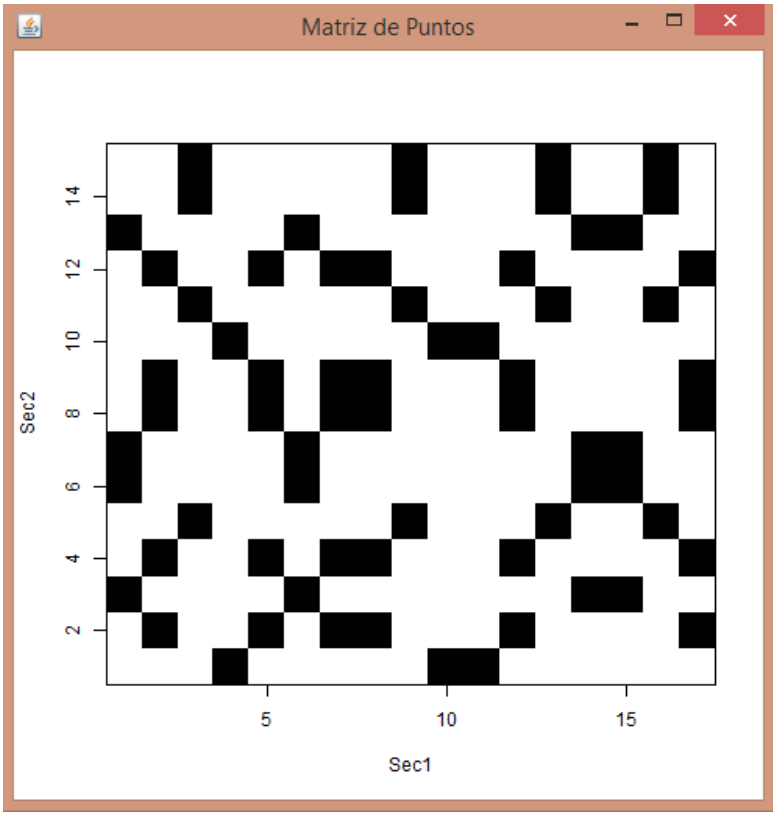


Figura 22. Matriz de puntos.

JADE Remote Agent Management GUI

Seleccionando el contenedor principal de los agentes "Main-Container" (y de los diferentes servicios como páginas amarillas y páginas blancas) y pulsando sobre el icono "Start Sniffer", se podrá realizar un seguimiento del paso de mensajes y de la comunicación existente entre los agentes a nivel interno de la aplicación.

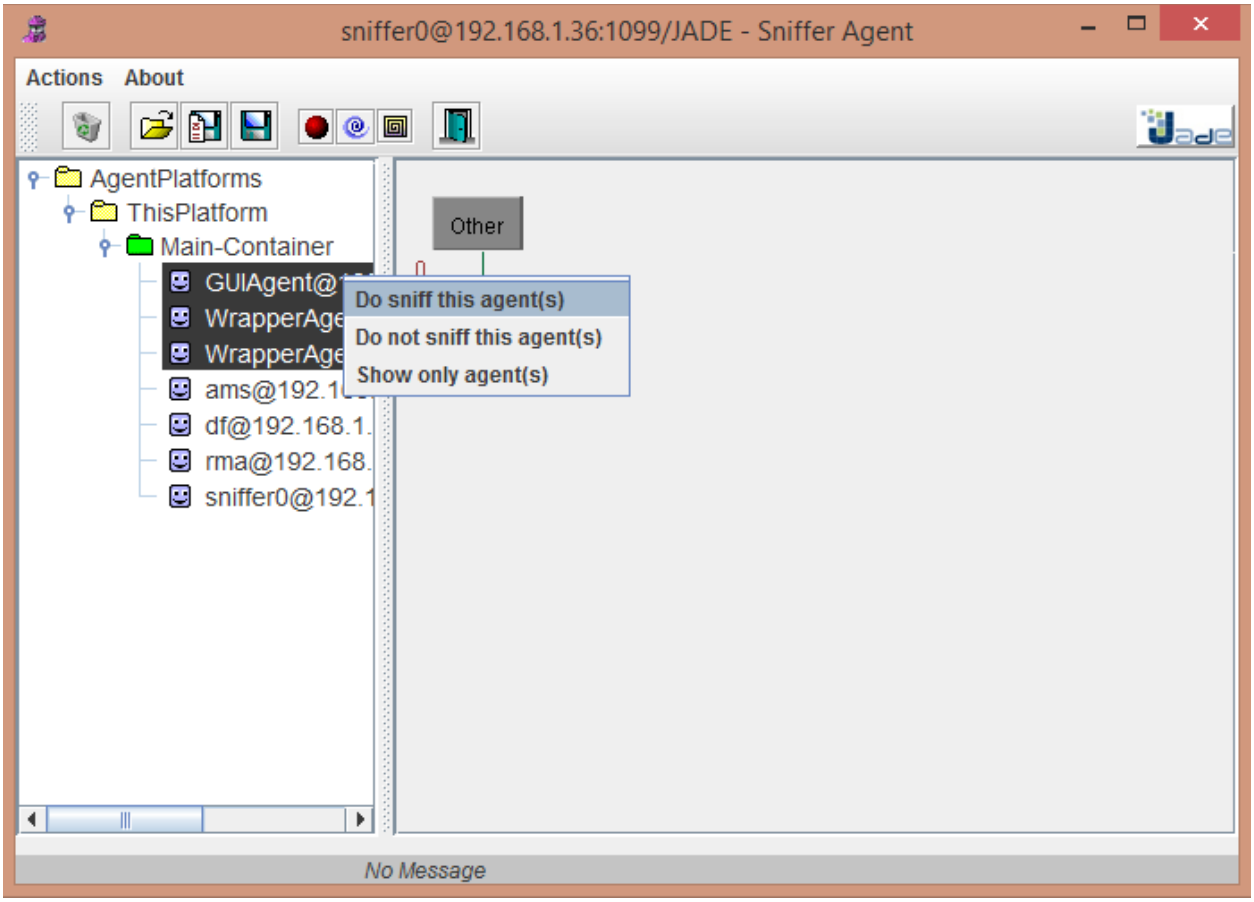


Figura 23. Sniffer para visualizar la comunicación entre agentes.

Una vez se haya pulsado sobre el botón del sniffer, se abrirá una nueva ventana desde donde se añadirán los agentes que queramos visualizar (Figura 23). Para ello, se seleccionan los agentes, se despliega un menú mediante el botón derecho del ratón, y se añaden los agentes a la interfaz pulsando sobre la opción "Do sniff this agent(s)".

Tras disparar el usuario un evento desde la interfaz, el agente *GUIAgent* solicita al servicio DF los agentes *WrapperAgent* que se encuentren disponibles. Tras elegir uno de ellos, le envía un mensaje con los parámetros del usuario almacenados. El agente *WrapperAgent* lo recibe, ejecuta el script correspondiente y se deregistra del servicio DF. Una vez finalizada la ejecución, el comportamiento asociado al script envía un mensaje al *WrapperAgent* (a sí mismo) con los resultados del alineamiento. Inmediatamente el *WrapperAgent* envía los resultados al

GUIAgent que se encargará de publicarlos en la interfaz. En ese momento, el *WrapperAgent* vuelve a registrarse en el servicio DF para poder recibir nuevas peticiones (Figura 24).

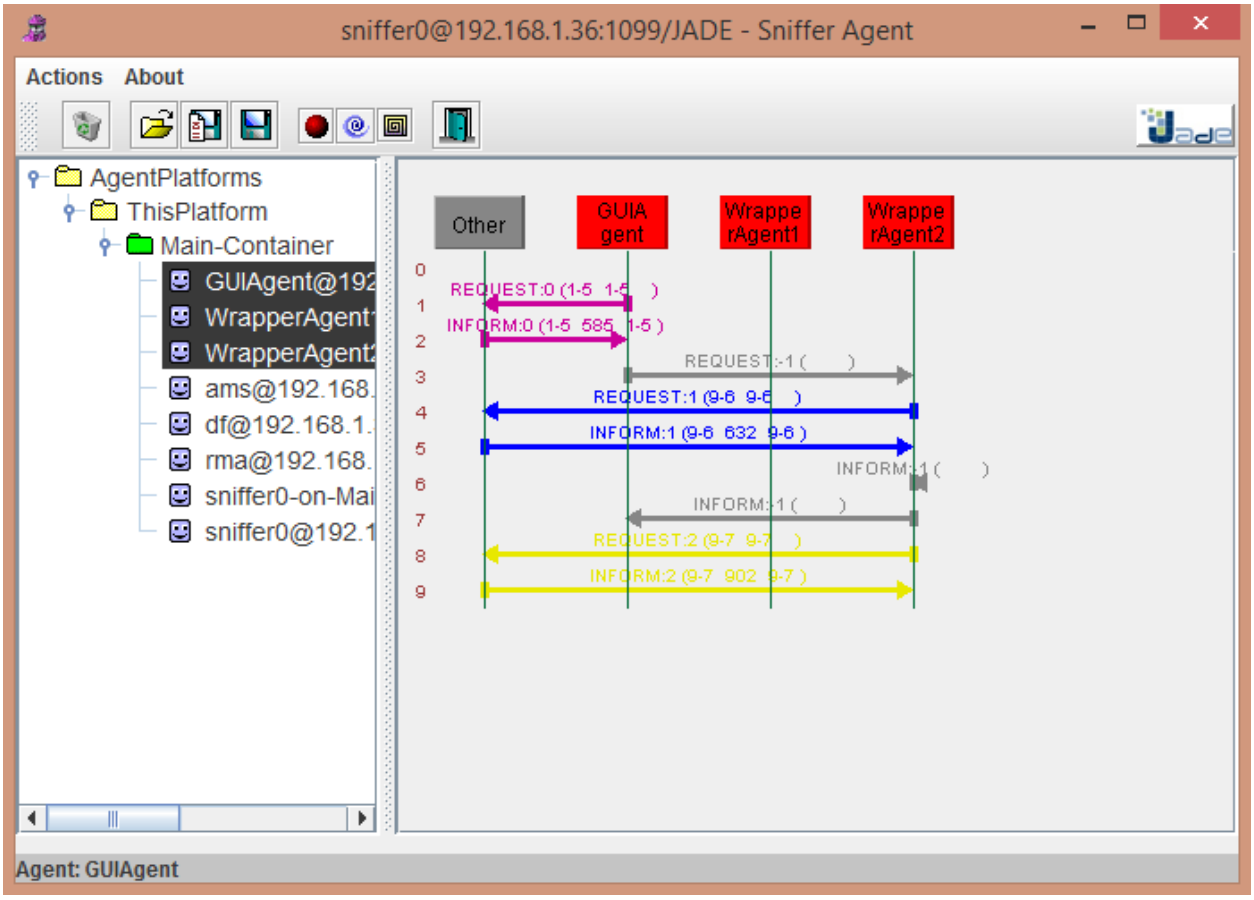


Figura 24. Comunicación entre agentes durante la ejecución utilizando el Sniffer.

7. Aportaciones de este trabajo

En este apartado se mostrarán una serie de ejemplos de ejecución a partir de los códigos desarrollados por el autor. La razón fundamental para su reescritura, ha sido el interés en analizar el comportamiento del algoritmo por partes, añadir otras funcionalidades y realizar comparativas de ejecución frente a otros desarrollos.

7.1. Alineamientos globales

El clásico algoritmo de alineamiento global por pares utilizando programación dinámica, es el algoritmo Needleman-Wunsch. En este algoritmo, un alineamiento óptimo es obtenido sobre la totalidad de las longitudes de las dos secuencias. Debe extenderse desde el principio hasta el final de ambas secuencias para conseguir la puntuación máxima. En otras palabras, la ruta de alineamiento tiene que ir desde la esquina inferior derecha de la matriz, hasta la esquina superior izquierda de la misma. El inconveniente de centrarse en obtener la puntuación máxima del alineamiento sobre la longitud completa de una secuencia, es el riesgo a perder la mejor similitud local. Esta estrategia sólo es adecuada para el alineamiento de dos secuencias estrechamente relacionadas que tienen la misma longitud. Para secuencias divergentes o secuencias con diferente dominio estructural, la aproximación no produce un alineamiento óptimo. Comenzaremos explicando las ecuaciones que utilizaremos como base para alcanzar dicho alineamiento entre dos secuencias:

$$\begin{aligned} S(0,0) &= 0 \\ S(i,0) &= \gamma\, i \\ S(0,j) &= \gamma\, j \end{aligned} \tag{38}$$

Para poder realizar la recursividad necesaria dentro de la matriz (S) de alineamientos parciales, será necesario introducir una fila y una columna auxiliar. En la posición (0,0) de la matriz tendremos un valor nulo, y en el resto de los índices de la fila (i) y de la columna (j) auxiliar, situaremos el valor resultante de la multiplicación entre el índice y el valor que le hayamos asignado a la penalización por hueco (γ).

$$S(i,j)=max\begin{cases} S(i-1,j-1)+\sigma(x_i,y_i), \\ S(i-1,j)+\gamma, \\ S(i,j-1)+\gamma. \end{cases} \quad (38)$$

La ecuación de recursividad, nos permitirá rellenar las celdas restantes de la matriz (S) de alineamientos parciales.

Para ello, nos situaremos en la celda que representa el primer índice de la fila (i) y de la columna (j), y escogeremos el valor máximo resultante de entre los siguientes valores:

- El valor resultante de la suma de la celda situada en la posición S(i-1,j-1) más el valor de los residuos correspondientes de la la matriz de sustitución.

- El valor resultante de la suma de la celda situada en la posición S(i-1,j) más el valor asignado a la penalización por hueco (*γ)*.

- El valor resultante de la suma de la celda situada en la posición S(i,j-1) más el valor asignado a la penalización por hueco (*γ*).

Una vez rellenada la matriz de alineamientos parciales, realizamos desde la posición M y N de la matriz, el 'tracing-back', para ir escogiendo los residuos (o los gaps) correspondientes para cada una de las secuencias hasta alcanzar la posición S(0,0) de la matriz de alineamientos parciales.

A continuación, mostraremos un ejemplo de ejecución de la función programada en R (ver apéndices 1 y 2) para calcular el alineamiento global por pares de dos secuencias de nucleótidos de ADN.

Las secuencias a alinear serán las siguientes:

- Secuencia 1: ATCGTATTCGGTCAACT
- Secuencia 2: GTATCAATTGCTACC

Una vez estemos en la consola de R, cargaremos el script mediante el comando `source("ruta del script")`. A continuación, ejecutaremos la función de cálculo introduciendo los siguientes parámetros:

NeedlemanWunsch<-function(Sec1,Sec2,Hueco,MSHeader,MSData)

- Sec1: Vector de la secuencia a alinear.
- Sec2: Vector de la secuencia a alinear.
- Hueco: Valor de la penalización por hueco.
- MSHeader: Vector de la cabecera de la matriz de puntuación a utilizar.
- MSData: Vector de los valores (sin comillas) de la matriz de puntuación rellenados por filas de izquierda a derecha y de arriba a abajo.

Si no le asignáramos ningún valor al campo Hueco, automáticamente el script le asignaría el valor 0. Debido a la sobrecarga de parámetros realizada con la inicialización de la matriz de puntuación, si dejáramos el campo `MSData` vacío, le asignaría automáticamente los valores de la matriz identidad.

En nuestro ejemplo, añadiremos las dos secuencias antes mencionadas y asignaremos un valor de -5 a la penalización por hueco; quedando la función de la siguiente manera:

```
NeedlemanWunsch(c("A","T","C","G","T","A","T","T","C","G",”G
","T","C","A","A","C","T"),c("G","T","A","T","C","A","A","T"
,"T","G","C","T","A","C","C"),-5,c("A","G","C","T"),c(10,-1,
-3,-4,-1,7,-5,-3,-3,-5,9,0,-4,-3,0,8))
```

Tras llamar a la función, ésta nos mostrará primeramente por pantalla las dos secuencias que le hemos proporcionado para alinear:

```
[1] "Sequence 1"
A T C G T A T T C G G T C A A C T
[1] "Sequence 2"
G T A T C A A T T G C T A C C
```

Posteriormente, aparecerá la matriz de sustitución o de puntuación para cada nucleótido de ADN con la que la función trabajará para rellenar la matriz de alineamientos parciales (ver Tabla 17):

Tabla 17. Matriz de sustitución o de puntuación para cada nucleótido de ADN.

	A	**G**	**C**	**T**
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

A continuación, observaremos la matriz de alineamientos parciales (junto con la fila y columna auxiliares) con las dos secuencias encabezando el eje horizontal y vertical (ver Tabla 18):

Tabla 18. Matriz de alineamientos parciales (junto con la fila y columna auxiliares) con las dos secuencias encabezando el eje horizontal y vertical.

		A	**T**	**C**	**G**	**T**	**A**	**T**	**T**	**C**	**G**	**G**	**T**	**C**	**A**	**A**	**C**	**T**
	0	-5	-10	-15	-20	-25	-30	-35	-40	-45	-50	-55	-60	-65	-70	-75	-80	-85
G	-5	-1	-6	-11	-8	-13	-18	-23	-28	-33	-38	-43	-48	-53	-58	-63	-68	-73
T	-10	-6	7	2	-3	0	-5	-10	-15	-20	-25	-30	-35	-40	-45	-50	-55	-60
A	-15	0	2	4	1	-4	10	5	0	-5	-10	-15	-20	-25	-30	-35	-40	-45
T	-20	-5	8	3	1	9	5	18	13	8	3	-2	-7	-12	-17	-22	-27	-32
C	-25	-10	3	17	12	7	6	13	18	22	17	12	7	2	-3	-8	-13	-18
A	-30	-15	-2	12	16	11	17	12	13	17	21	16	11	6	12	7	2	-3
A	-35	-20	-7	7	11	12	21	16	11	12	16	20	15	10	16	22	17	12
T	-40	-25	-12	2	6	19	16	29	24	19	14	15	28	23	18	17	22	25
T	-45	-30	-17	-3	1	14	15	24	37	32	27	22	23	28	23	18	17	30
G	-50	-35	-22	-8	4	9	13	19	32	32	39	34	29	24	27	22	17	25
C	-55	-40	-27	-13	-1	4	8	14	27	41	36	34	34	38	33	28	31	26
T	-60	-45	-32	-18	-6	7	3	16	22	36	38	33	42	37	34	29	28	39
A	-65	-50	-37	-23	-11	2	17	12	17	31	35	37	37	39	47	44	39	34
C	-70	-55	-42	-28	-16	-3	12	17	12	26	30	32	37	46	42	44	53	48
C	-75	-60	-47	-33	-21	-8	7	12	17	21	25	27	32	46	43	39	53	53

Después se nos mostrará el alineamiento óptimo de las dos secuencias:

```
[1] "Sequence 1 Optimal Alignment"
- - A T C G T A T T C G G T C A A C T

[1] "Sequence 2 Optimal Alignment"
G T A T C A - A T T - - G - C T A C C
```

Para continuar con la visualización de la puntuación del alineamiento óptimo:

```
[1] "Optimal Alignment Score"
53
```

Y ya, para finalizar, se mostrará el número de huecos (gaps) utilizados para realizar el alineamiento:

```
[1] "Number of Gaps"
6
```

No debemos olvidarnos de la ventana gráfica de R que aparecerá para mostrarnos el método gráfico por puntos utilizado con el fin de ayudarnos a identificar las regiones con mayor similitud (ver Figura 25). En ella podemos observar la diagonal afectada por las inserciones o deleciones que dan lugar a huecos (gap) en forma de quiebros o zig-zageos. Los valores numéricos de los ejes X e Y, representan la posición de los residuos de las secuencias iniciales antes de ser alineadas. En este caso, el eje X representaría a los residuos de la primera secuencia, mientras que el eje Y representaría a los residuos de la segunda secuencia. Para dibujar el gráfico, se ha utilizado la función `dotPlot()` que está disponible dentro del paquete "S*equinR*".

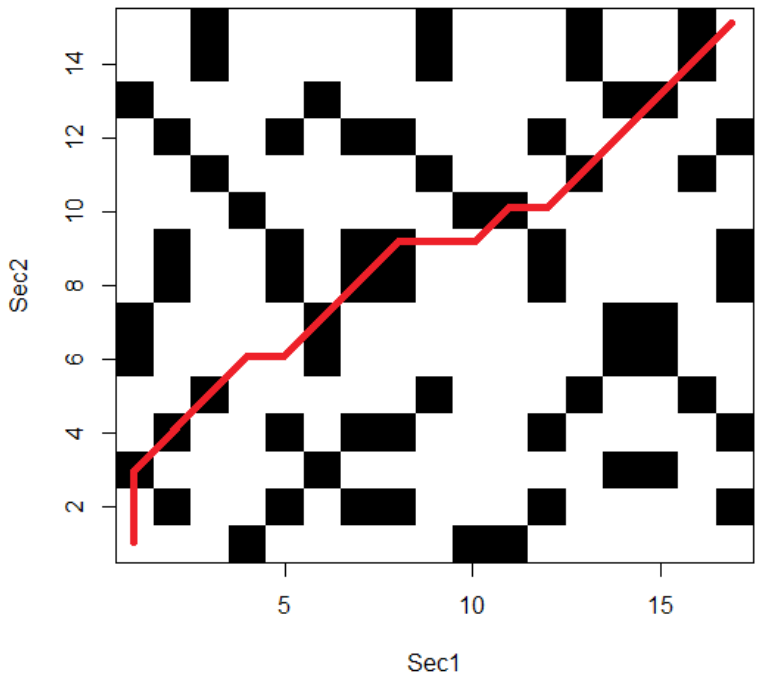


Figura 25. Ventana gráfica de R que aparecerá para mostrarnos el método gráfico por puntos utilizado con el fin de ayudarnos a identificar las regiones con mayor similitud.

Con la finalidad de que el cómputo sea más rápido, se ha modificado el script inicial para que tenga acceso a una DLL programada en C. Esta librería se encargará de agilizar el proceso de la asignación de valores en la matriz de alineamientos parciales.

El script en R modificado, se ha desarrollado con la única diferencia de que la matriz de alineamientos parciales se calculará en la DLL mediante la función `.C()`. Esta función será llamada una vez cargada nuestra librería dinámica con la función `dyn.load()` y descargada con `dyn.unload()` una vez terminada la ejecución de nuestra librería. Es importante saber que la función `.C()` es incompatible con el tipo `matrix`, por lo que deberíamos convertir el argumento a vector utilizando `as.vector()` e indicar en nuestro caso que es un vector de enteros `as.integer()`. Debido a la alta complejidad que conlleva la depuración de errores en librerías dinámicas, se ha optado por no utilizar bibliotecas en nuestra implementación en lenguaje C (se explicará al detalle más adelante). Es por ello que se ha decidido asociar un valor numérico a cada tipo de valor alfanumérico de la cabecera de la matriz de puntuación `MSHeader()`, para después asociar un vector de enteros a cada secuencia `S1` y `S2` dependiendo de los elementos que lo conformen. Todos estos nuevos objetos (`MSHeaderN`, `S1N` y `S2N`) serán enviados a la DLL para utilizarlos en el cálculo de la matriz de alineamientos parciales. Es importante recordar que el primer parámetro de la función `.C()` es el nombre que le asignaremos, siendo el resto de argumentos partes de la función de llamada. En nuestro caso, de todos esos argumentos que nos devolverá la librería dinámica al terminar su ejecución, nos quedaremos con el vector `as.vector(as.integer(MatrizAlin))` indicándole que lo almacene en el objeto `resultado` que será el que finalmente se almacene en `MatrizAlin`. Para terminar, no hay que olvidar que hay que transformar los objetos del tipo `vector` devueltos por la librería dinámica que nos interese al tipo `matrix` para poder terminar con la ejecución restante del script en R.

La función programada en C y compilada sobre una DLL, es llamada desde R y le es pasada una serie de argumentos para posibilitar el cálculo y la asignación de valores sobre la matriz de alineamientos parciales. Una vez terminado el cómputo, la ejecución regresa a R devolviendo la matriz de alineamiento calculada. La función `NW()` recibe los argumentos que

son almacenados en variables con su respectiva posición en memoria (punteros). Es importante recordar que debido a la imposibilidad de enviar objetos del tipo `matrix` a la librería dinámica, será necesario convertir los vectores que creamos convenientes en nuestra DLL. Al recibir los vectores con valores numéricos `MSHeaderN`, `S1N` y `S2N` desde R, podemos prescindir de la biblioteca `string.h` y de su función para la comparación de caracteres `strcmp()`. También se ha prescindido de la biblioteca `math.h` y de su función `fmax()` utilizada para comparar dos valores del tipo `float` y devolver el valor más grande, ahorrándonos de esta manera la conversión de las variables `integer` a `float`. Para suplir esa carencia, se han programado una serie de sentencias condicionales que han posibilitado la comparación de tres variables `integer` de una manera mucho más simple y fácil, y sin tener que lidiar con la depuración de errores que supone trabajar con librerías.

```
May = Diag;

if (Arriba > May)
{
    May = Arriba;
}
if (Izq > May)
{
    May = Izq;
}

ma[i][j] = May;
```

Para finalizar, no hay que olvidar que será necesario volver a convertir las variables del tipo `matrix` a `vector` antes de enviarlas de nuevo a R.

7.2. Alineamientos locales

En los alineamientos de secuencias regulares, el nivel de divergencia entre dos secuencias para ser alineadas no son fácilmente conocidas. Las longitudes de las dos secuencias pueden ser desiguales y, en algunos casos, la identificación de similitudes en secuencias regionales pueden ser de mayor importancia que la búsqueda de coincidencias que incluye a todos los residuos. Además, los alineamientos locales cuentan con la ventaja de que detectan mejor las relaciones remotas que se han conservado en el tiempo. La primera aplicación de la programación dinámica en el alineamiento local es el algoritmo Smith-Waterman, donde las puntuaciones positivas son asignadas para el emparejamiento de residuos y cero para los disparejos; sin utilizar puntuaciones negativas y utilizando un procedimiento similar al rastreo hacia el origen (tracing-back) usado en la programación dinámica. Sin embargo, a diferencia del alineamiento global, la ruta de alineamiento puede comenzar y terminar internamente a lo largo de la diagonal principal. Primeramente, comenzaría en la posición que tenga la mayor puntuación; avanzando hacia arriba diagonalmente hasta alcanzar una celda que contenga un valor nulo, e introduciendo huecos si fuesen necesarios (utilizando a menudo la penalización por hueco afín). En ocasiones, se obtienen varios segmentos óptimamente alineados con mayor puntuación. Al igual que en el alineamiento global, el resultado final se verá influenciado por la elección de los sistemas de puntuación utilizados.

El objetivo del alineamiento local es conseguir la máxima puntuación, que puede estar a expensas de la puntuación global más alta para un alineamiento de longitud completa. Este enfoque puede ser el adecuado para el alineamiento divergente de secuencias o secuencias con múltiples dominios que pueden ser de diferentes orígenes.

Comenzaremos explicando las ecuaciones que utilizaremos como base para alcanzar el alineamiento local entre dos secuencias:

$$\begin{aligned} H(i,0)&=0, 0\leqslant i\leqslant m \\ H(0,j)&=0, 0\leqslant j\leqslant n \end{aligned} \tag{39}$$

Introduciremos en las celdas de la fila y de la columna auxiliar de la matriz de alineamientos parciales (H) el valor nulo para poder aplicar la ecuación de recurrencia.

$$H(i,j)=max\begin{Bmatrix} 0, \\ H(i-1,j-1)+\omega(a_i,b_j), \\ H(i-1,j)+\omega(a_i,-), \\ H(i,j-1)+\omega(-,b_j). \end{Bmatrix}, 1\leqslant i\leqslant m, 1\leqslant j\leqslant n \tag{39}$$

La ecuación de recurrencia nos permite rellenar la matriz de alineamientos parciales, se basa en ir rellenando las celdas desde la posición H(1,1) hasta la posición H(m,n). Nos situamos en la celda que se encuentra en la posición H(i,j) y escogeremos el valor máximo resultante de entre una serie de valores:

- El valor resultante de la suma de la celda situada en la posición H(i-1,j-1) más el valor de los residuos correspondientes de la la matriz de sustitución.

- El valor resultante de la suma de la celda situada en la posición H(i-1,j) más el valor asignado a la penalización por hueco.

- El valor resultante de la suma de la celda situada en la posición H(i,j-1) más el valor asignado a la penalización por hueco.

Tras haber rellenado la matriz de alineamientos parciales, es momento de realizar el 'tracing-back' desde la celda con el valor mayor H(i,j). Construiremos desde esa celda el

alineamiento local de las dos secuencias dependiendo el camino recorrido al rellenar la matriz de alineamientos parciales:

- Si se ha escogido la posición H(i-1,j-1), se añadirán a las secuencias finales $\omega(a_i,b_j)$ el residuo correspondiente a los índices de esa celda.

- Si se ha escogido la posición H(i-1,j), se añadirá a la secuencia final (a) el residuo correspondiente al índice de esa celda $\omega(a_i,-)$ y un hueco (gap) a la secuencia final (b).

- Si se ha escogido la posición H(i,j-1), se añadirá a la secuencia final (a) un hueco (gap) y el residuo correspondiente al índice de esa celda $\omega(-,b_j)$ a la secuencia final (b).

El proceso de 'tracing-back' continuará hasta alcanzar la celda H(0,0) o, por el contrario, la celda que contenga un valor nulo.

A continuación, mostraremos un ejemplo de ejecución de la función programada en R (ver apéndices 3 y 4) para calcular el alineamiento local por pares de dos secuencias de nucleótidos de ADN.

Las secuencias a alinear serán las siguientes:

- Secuencia 1: ATCGTATTCGGTCAACT
- Secuencia 2: GTATCAATTGCTACC

Una vez estemos en la consola de R, cargaremos el script mediante el comando `source("ruta del script")`. A continuación, ejecutaremos la función de cálculo introduciendo los siguientes parámetros:

SmithWaterman<-function(Sec1,Sec2,Hueco,MSHeader,MSData)

- Sec1: Vector de la secuencia a alinear.
- Sec2: Vector de la secuencia a alinear.
- Hueco: Valor de la penalización por hueco.
- MSHeader: Vector de la cabecera de la matriz de puntuación a utilizar.
- MSData: Vector de los valores (sin comillas) de la matriz de puntuación rellenados por filas de izquierda a derecha y de arriba a abajo.

Si no le asignáramos ningún valor al campo Hueco, automáticamente el script le asignaría el valor 0. Debido a la sobrecarga de parámetros realizada con la inicialización de la matriz de puntuación, si dejáramos el campo `MSData` vacío, le asignaría automáticamente los valores de la matriz identidad.

En nuestro ejemplo, añadiremos las dos secuencias antes mencionadas y asignaremos un valor de -5 a la penalización por hueco; quedando la función de la siguiente manera:

```
SmithWaterman(c("A","T","C","G","T","A","T","T","C","G","G",
"T","C","A","A","C","T"),c("G","T","A","T","C","A","A","T","
T","G","C","T","A","C","C"),-5,c("A","G","C","T"),c(10,-1,-
3,-4,-1,7,-5,-3,-3,-5,9,0,-4,-3,0,8))
```

Tras llamar a la función, ésta nos mostrará primeramente por pantalla las dos secuencias que le hemos proporcionado para alinear:

```
[1] "Sequence 1"
A T C G T A T T C G G T C A A C T

[1] "Sequence 2"
G T A T C A A T T G C T A C C
```

Posteriormente, aparecerá la matriz de sustitución o de puntuación para cada nucleótido de ADN con la que la función trabajará para rellenar la matriz de alineamientos parciales (ver Tabla 19):

Tabla 19. Matriz de sustitución o de puntuación para cada nucleótido de ADN.

	A	**G**	**C**	**T**
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

A continuación, observaremos la matriz de alineamientos parciales (junto con la fila y columna auxiliares) con las dos secuencias encabezando el eje horizontal y vertical (ver Tabla 20):

Tabla 20. Matriz de alineamientos parciales (junto con la fila y columna auxiliares) con las dos secuencias encabezando el eje horizontal y vertical.

		A	**T**	**C**	**G**	**T**	**A**	**T**	**T**	**C**	**G**	**G**	**T**	**C**	**A**	**A**	**C**	**T**
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	7	2	0	0	0	0	7	7	2	0	0	0	0	0
T	0	0	8	3	2	15	10	8	8	3	2	7	15	10	5	0	0	8
A	0	10	5	8	3	10	25	20	15	10	5	2	10	15	20	15	10	5
T	0	5	18	13	8	11	20	33	28	23	18	13	10	10	15	20	15	18
C	0	0	13	27	22	17	15	28	33	37	32	27	22	19	14	15	29	24
A	0	10	8	22	27	22	27	23	28	33	37	32	27	22	29	24	24	29
A	0	10	10	17	22	27	32	27	23	28	33	37	32	27	32	39	34	29
T	0	5	18	13	17	30	27	40	35	30	28	33	45	40	35	34	39	42
T	0	0	13	18	13	25	30	35	48	43	38	33	41	45	40	35	34	47
G	0	0	8	13	25	20	25	30	43	48	50	45	40	41	45	40	35	42
C	0	0	3	17	20	25	38	25	38	52	48	50	45	49	44	45	49	44
T	0	0	8	12	17	28	25	28	33	47	52	48	58	53	49	44	45	57
A	0	10	5	8	12	23	38	33	28	42	47	52	53	58	63	59	54	52

C	0	5	10	14	9	18	33	38	33	37	42	47	52	62	58	63	68	63
C	0	0	5	19	14	13	28	33	38	42	37	42	47	61	62	58	72	68

Después se nos mostrará el alineamiento óptimo de las dos secuencias:

```
[1] "Sequence 1 Optimal Alignment"
A T C G T A T T C G G T C A A C

[1] "Sequence 2 Optimal Alignment"
A T C - A A T T - G C T - A C C
```

Para continuar con la visualización de la puntuación del alineamiento óptimo:

```
[1] "Optimal Alignment Score"
72
```

Y ya, para finalizar, se mostrará el número de huecos (gaps) utilizados para realizar el alineamiento:

```
[1] "Number of Gaps"
3
```

No debemos olvidarnos de la ventana gráfica de R que aparecerá para mostrarnos el método gráfico por puntos utilizado con el fin de ayudarnos a identificar las regiones con mayor similitud (ver Figura 26). En ella podemos observar la diagonal afectada por las inserciones o deleciones que dan lugar a huecos (gap) en forma de quiebros o zig-zageos. Los valores numéricos de los ejes X e Y, representan la posición de los residuos de las secuencias iniciales antes de ser alineadas. En este caso, el eje X representaría a los residuos de la primera secuencia, mientras que el eje Y representaría a los residuos de la segunda secuencia. Para dibujar el gráfico, se ha utilizado la función `dotPlot()` que está disponible dentro del paquete "S*equinR"*.

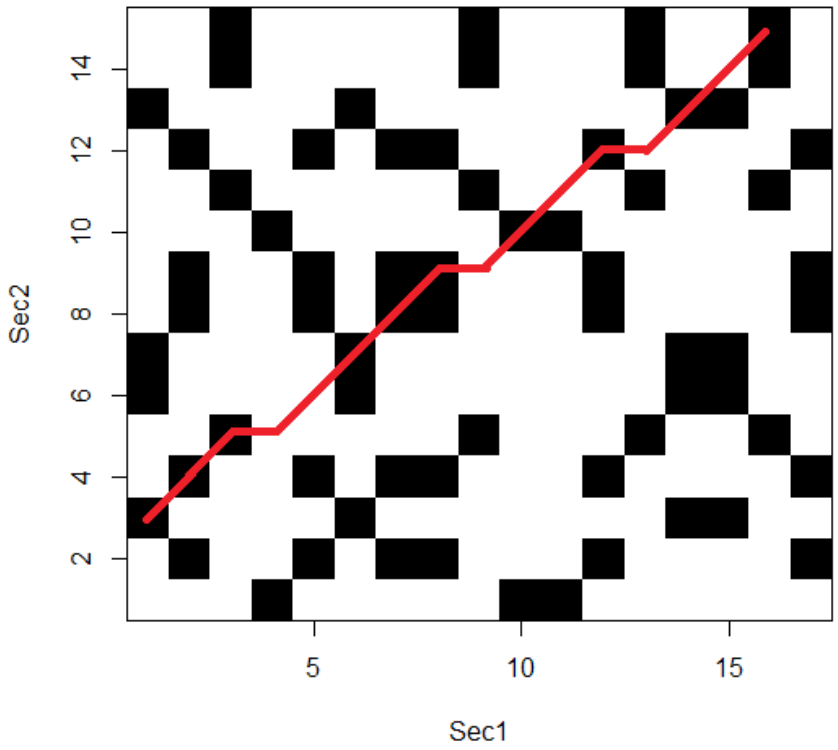


Figura 26. Ventana gráfica de R que aparecerá para mostrarnos el método gráfico por puntos utilizado con el fin de ayudarnos a identificar las regiones con mayor similitud.

Con la finalidad de que el cómputo sea más rápido, se ha modificado el script inicial para que tenga acceso a una DLL programada en C. Esta librería se encargará de agilizar el proceso en la asignación de valores en la matriz de alineamientos parciales.

El script en R modificado, se ha desarrollado con la única diferencia de que la matriz de alineamientos parciales se calculará en la DLL mediante la función `.C()`. Esta función será llamada una vez cargada nuestra librería dinámica con la función `dyn.load()` y descargada con `dyn.unload()` una vez terminada la ejecución de nuestra librería. Es importante saber que la función `.C()` es incompatible con el tipo `matrix`, por lo que deberíamos convertir el argumento a vector utilizando `as.vector()` e indicar en nuestro caso que es un vector de enteros `as.integer()`. Debido a la alta complejidad que conlleva la depuración de errores en librerías dinámicas, se ha optado por no utilizar bibliotecas en nuestra implementación en

lenguaje C (se explicará al detalle más adelante). Es por ello que se ha decidido asociar un valor numérico a cada tipo de valor alfanumérico de la cabecera de la matriz de puntuación `MSHeader()`, para después asociar un vector de enteros a cada secuencia `S1` y `S2` dependiendo de los elementos que lo conformen. Todos estos nuevos objetos (`MSHeaderN`, `S1N` y `S2N`) serán enviados a la DLL para utilizarlos en el cálculo de la matriz de alineamientos parciales. Es importante recordar que el primer parámetro de la función `.C()` es el nombre que le asignaremos, siendo el resto de argumentos partes de la función de llamada. En nuestro caso, de todos esos argumentos que nos devolverá la librería dinámica al terminar su ejecución, nos quedaremos con el vector `as.vector(as.integer(MatrizAlin))` indicándole que lo almacene en el objeto `resultado` que será el que finalmente se almacene en `MatrizAlin.` Para terminar, no hay que olvidar que hay que transformar los objetos del tipo `vector` devueltos por la librería dinámica que nos interese al tipo `matrix` para poder terminar con la ejecución restante del script en R.

La función programada en C y compilada sobre una DLL, es llamada desde R y le es pasada una serie de argumentos para posibilitar el cálculo y la asignación de valores sobre la matriz de alineamientos parciales. Una vez terminado el cómputo, la ejecución regresa a R devolviendo la matriz de alineamiento calculada. La función `SW()` recibe los argumentos que son almacenados en variables con su respectiva posición en memoria (punteros). Es importante recordar que debido a la imposibilidad de enviar objetos del tipo `matrix` a la librería dinámica, será necesario convertir los vectores que creamos convenientes en nuestra DLL. Al recibir los vectores con valores numéricos `MSHeaderN`, `S1N` y `S2N` desde R, podemos prescindir de la biblioteca `string.h` y de su función para la comparación de caracteres `strcmp()`. También se ha prescindido de la biblioteca `math.h` y de su función `fmax()` utilizada para comparar dos valores del tipo `float` y devolver el valor más grande, ahorrándonos de esta manera la conversión de las variables `integer` a `float`. Para suplir esa carencia, se han programado una serie de sentencias condicionales que han posibilitado la comparación de tres variables `integer` de una manera mucho más simple y fácil, y sin tener que lidiar con la depuración de errores que supone trabajar con librerías.

```
May = Diag;

if (Arriba > May)
{
    May = Arriba;
}
if (Izq > May)
{
    May = Izq;
}

ma[i][j] = May;
```

Para finalizar, no hay que olvidar que será necesario volver a convertir las variables del tipo `matrix` a `vector` antes de enviarlas de nuevo a R.

7.3. Resultados comparativos

En las siguientes pruebas, se comprobará la eficiencia y potencial de cómputo de los scripts en R que acceden a funciones contenidas en DLLs programadas en C, frente a los scripts convencionales programados en R. Para ello, alinearemos 3 conjuntos de pares de secuencias de ADN y compararemos el tiempo de cómputo que registraremos como resultado. En las 3 pruebas que se efectuarán, se alinearán pares de cadenas de la misma longitud: de 500, de 1000 y de 2000 caracteres. Estos datos se han generado de forma aleatoria como datos simulados para la realización de esta prueba, sin pérdida de generalidad.

Para calcular el tiempo de ejecución, insertaremos las siguientes líneas de código en nuestros scripts. Se recogerá el tiempo del sistema al iniciar y al finalizar la ejecución del script. Posteriormente, se realizará la diferencia de esos valores para conocer el tiempo real de la ejecución:

```
start.time <- Sys.time()

end.time <- Sys.time()
time.taken <- start.time - end.time
time.taken
```

Con los resultados que se muestran a continuación (ver Tabla 21, Figura 27, Tabla 22 y Figura 28), se comprueba que realmente los scripts que acceden a librerías para realizar cálculos, son más eficientes en términos de velocidad que los que realizan todo el cómputo sobre R.

Tabla 21. Tabla de tiempos con los resultados de ejecución de los scripts en R con y sin DLL para los alineamientos globales.

Longitud cadenas	Script R (seg)	Script R con DLL (seg)
500	9	1
1000	34	3
2000	147	12

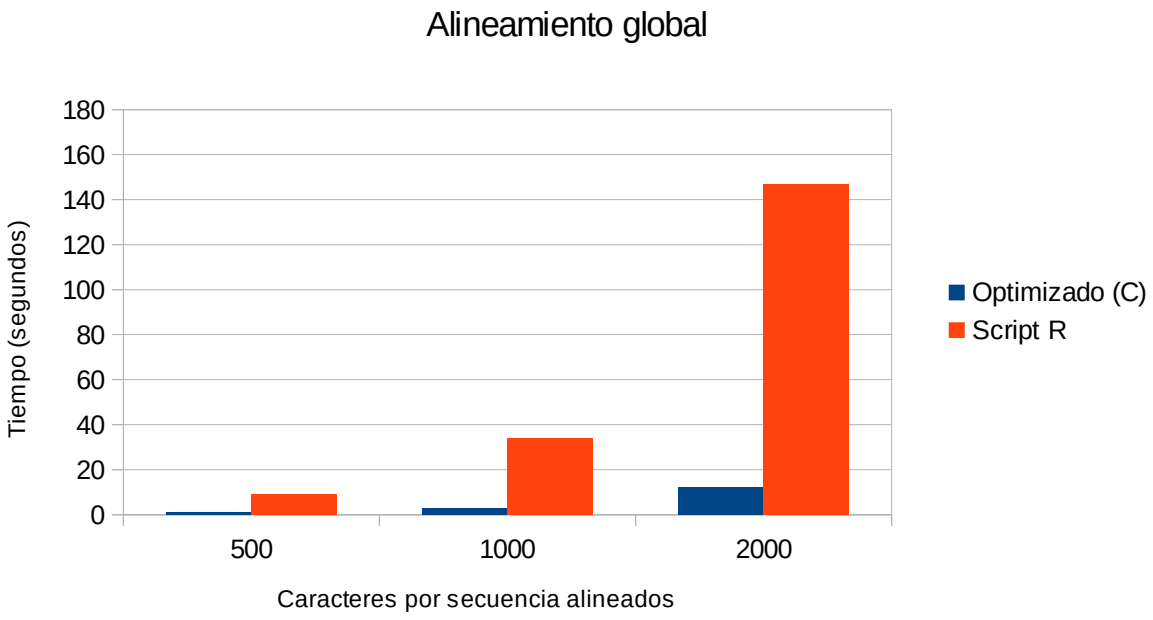


Figura 27. Gráfica con los tiempos de ejecución de los scripts en R con y sin DLL para los alineamientos globales.

Tabla 22. Tabla de tiempos con los resultados de ejecución de los scripts en R con y sin DLL para los alineamientos locales.

Longitud cadenas	Script R (seg)	Script R con DLL (seg)
500	29	5
1000	108	8
2000	438	43

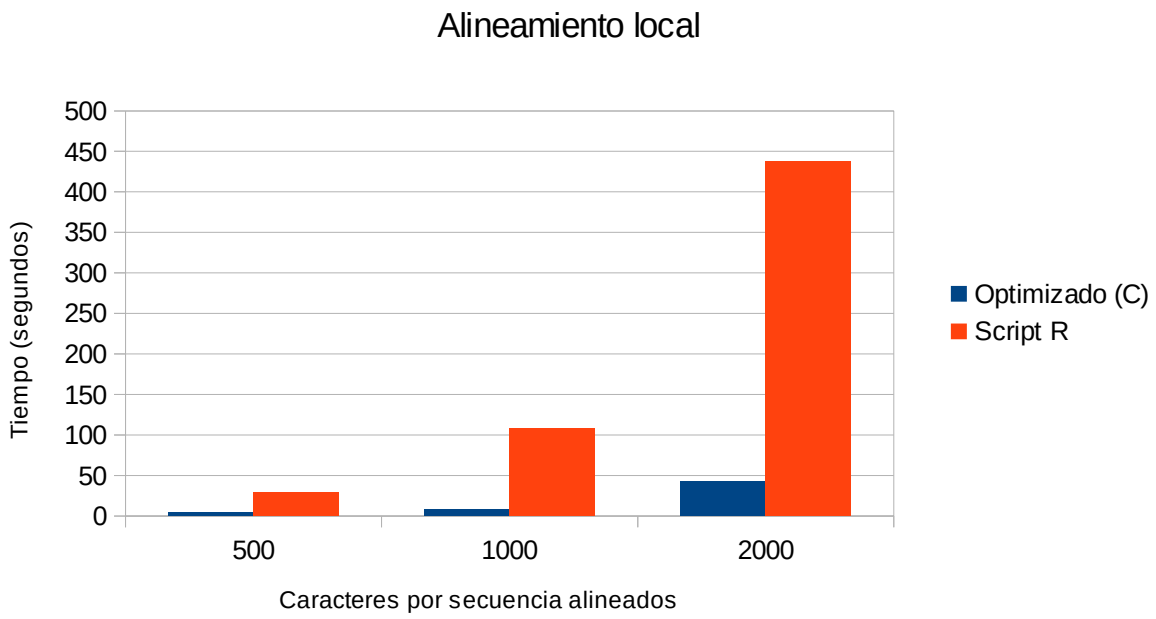


Figura 28. Gráfica con los tiempos de ejecución de los scripts en R con y sin DLL para los alineamientos locales.

Como se observa en las tablas y en los gráficos, se ha ejecutado el algoritmo para cadenas de longitudes 500, 1000 y 2000 elementos. Los tiempos de ejecución expresados en segundos se muestran en el eje Y, mientras que las longitudes utilizadas se muestran en el eje X; siendo el color azul el correspondiente para los scripts optimizados con librerías dinámicas y el rojo para los scripts desarrollados únicamente en R. Es destacable comentar que aunque los alineamientos locales sean ligeramente más lentos que los globales, sigue existiendo una gran diferencia entre los scripts en R y los scripts optimizados con librerías dinámicas, como bien se puede visualizar en las gráficas. En los alineamientos globales, encontramos que para las cadenas de 500 caracteres, el script en R tiene una duración de ejecución de 9 segundos (nueve veces más

lento que el script optimizado). Para las cadenas de 1000 caracteres, hay una diferencia de tiempo de ejecución de 31 segundos entre el script en R y el optimizado, que asciende a 135 segundos en el caso de las secuencias de 2000 elementos. En los alineamientos locales, en cambio, encontramos que para las cadenas de 500 caracteres, el script en R tiene una duración de ejecución de 29 segundos frente a los 5 segundos del script con DLL. Para las cadenas de 1000 caracteres, hay una diferencia de tiempo de ejecución de 100 segundos entre el script en R y el optimizado, que asciende a 395 segundos en el caso de las secuencias de 2000 elementos. No se han añadido comparativas de cadenas más largas debido al gran consumo de memoria de los algoritmos programados y a la limitada memoria disponible en el equipo donde se han realizado las pruebas; lo que ha provocado la aparición de mensajes en los scripts en R informando de memoria insuficiente, y la terminación automática de las ejecuciones en los scripts optimizados.

Cabe mencionar, que Bioconductor dispone de paquetes que permiten la secuenciación de datos utilizando algoritmos globales y locales. El paquete *"Biostrings"*, por ejemplo, dispone de la función `pairwiseAlignment()` basada en la técnica de la Programación Dinámica para el alineamiento de secuencias. Al igual que nuestros algoritmos, el consumo de memoria y tiempo de computación va en proporción al producto de la longitud de las secuencias a alinear. A continuación se presenta una tabla de tiempos [40] de cadenas alineadas globalmente por pares con la función antes mencionada (ver Tabla 23 y Figura 29):

Tabla 23. Tabla de tiempos con los resultados de ejecución de la función pairwiseAlignment() del paquete Biostrings utilizando cadenas de diferente longitud alineadas globalmente.

Longitud cadenas	500	1000	2000
Tiempo (seg)	0.024	0.048	0.136

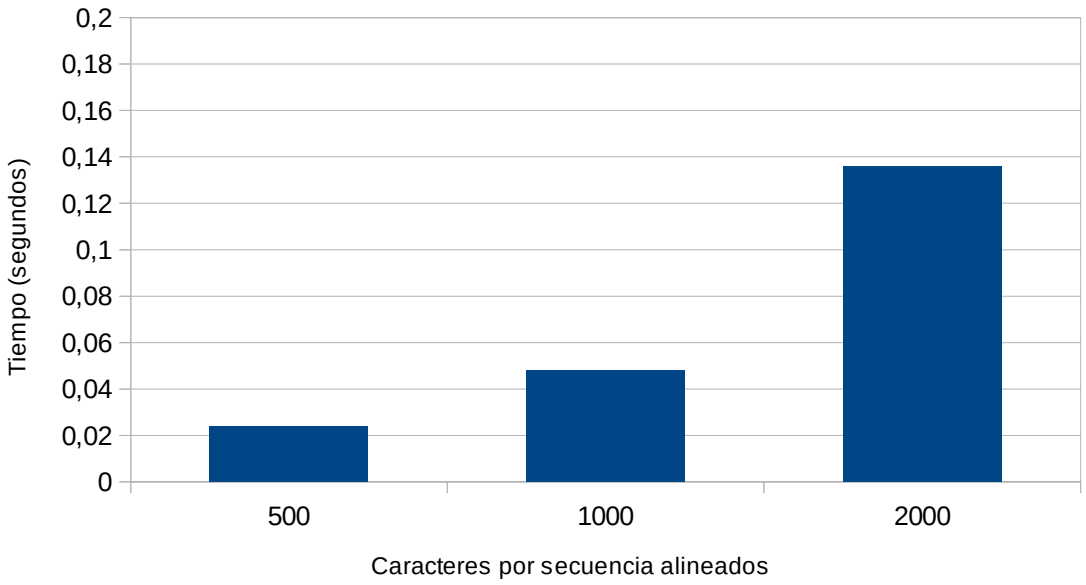


Figura 29. Gráfica con los tiempos de ejecución de la función pairwiseAlignment() del paquete Biostrings utilizando cadenas de diferente longitud alineadas globalmente.

Una de las principales diferencias con las funciones que se han programado, es que *"Biostrings"* utiliza cadena de caracteres en vez de vectores para sus alineamientos. Además sería importante anotar que para la construcción y visualización de las matrices por puntos, es necesario un tiempo considerable de cómputo que no viene reflejado en las pruebas realizadas en *"Biostrings"* basadas únicamente en el alineamiento. Sería interesante también reprogramar la totalidad de los algoritmos sobre librerías dinámicas y no sólo la matriz de alineamientos parciales como se ha realizado en este trabajo, con el fin de agilizar aún más la velocidad de cómputo y tener resultados todavía más cercanos a la tabla de tiempos obtenida en los alineamientos sobre la librería *"Biostrings"*.

8. Conclusiones y trabajo futuro

En este trabajo se han analizado y desarrollado mejoras de diseño y de ejecución sobre algoritmos de alineamiento de secuencias con la técnica de Programación Dinámica. La inclusión de características de sobrecarga en los parámetros ha permitido la penalización por gaps y la inclusión de características de sobrecarga en los parámetros que han permitido no sólo la penalización por gaps, sino también la utilización opcional de matrices de puntuación (PAM y BLOSOM) de aminoácidos permitiendo optimizar la valoración de los alineamientos. Además, se han desarrollado alternativas de programación de los algoritmos mostrados mediante el uso de librerías dinámicas desarrolladas en lenguaje C. Este desarrollo alternativo ha servido para comprobar la mejora del rendimiento que, como se ha demostrado en este trabajo, ha podido obtenerse con la compilación de determinadas partes del código en los distintos lenguajes.

Aprovechando la facilidad del lenguaje R para utilizar rutinas de otros lenguajes, se ha conseguido un gran avance en términos de eficiencia en el análisis y secuenciación de datos genómicos de alto rendimiento. Las llamadas a funciones compiladas sobre un archivo externo en forma de librería dinámica, ha permitido aprovechar todo el potencial que posee el lenguaje compilado C en la resolución de problemas con alto coste computacional. Todas las ventajas de los lenguajes compilados y las carencias de los lenguajes interpretados, han quedado reflejadas en las gráficas y tablas de tiempos calculadas sobre los algoritmos de alineamiento Needleman-Wunsh (global) y Smith-Waterman (local) donde hemos encontrado una diferencia de rendimiento considerable.

Además, el desarrollo del sistema multi-agente "MASBioseq" ha permitido incorporar mejoras de interfaz gráfica para la paralelización de procesos y la elección de máquinas donde estos se ejecutan.

Este trabajo ha sido incluido en el proyecto "Bioseq: Una librería R para el análisis de secuencias de datos" y presentado en el CEDI 2013. En él, se ofrece una librería orientada al tratamiento de bases de datos para que usuarios, profesores e investigadores puedan utilizarla

para el análisis de tablas y secuencias de datos [41], [42].

La continuidad de este trabajo como futura tesis doctoral, estará centrado en el desarrollo de algoritmos múltiples de alineamiento mejorados implementando técnicas de soft computing y lógica difusa. La lógica difusa se basa en la teoría de conjuntos difusos propuesto por L.A. Zadeh en 1965 [43], [44] que no es más que una forma de lógica multivaluada que trata el razonamiento aproximado en vez de fijo y exacto. En la "lógica clásica", poniendo como ejemplo la lógica booleana, los conjuntos binarios tienen dos posibles valores lógicos: verdadero o falso. En las variables de la lógica difusa, en cambio, pueden tener un valor de verdad que varía gradualmente entre 0 y 1. La lógica difusa se ha ampliado para manejar el concepto de verdad parcial, donde el valor de verdad pueda oscilar entre completamente cierto y totalmente falso [45].

En un sistema difuso, los valores de entrada fusificados ejecutan todas las reglas de un repositorio de conocimiento que tenga como parte de su premisa la entrada fusificada. Este proceso genera un nuevo conjunto difuso representando cada variable o solución de salida. La defusificación crea un valor para la variable de salida de ese nuevo conjunto difuso [46]. Por lo tanto, con el fin de aplicar la lógica difusa a una aplicación, primero los valores de entrada deben ser fusificados de modo que su valor esté entre un rango de 0 a 1, siendo entonces cuando se aplican las reglas definidas por la aplicación. Después de esto, los resultados derivados de varias reglas se combinan usando una función de agregación, para luego, defusificar los resultados agregados utilizando una función de inferencia. Las evaluaciones de las reglas difusas y la combinación de los resultados de las reglas individuales se realizan mediante operaciones de conjuntos difusos, siendo las operaciones sobre conjuntos difusos diferentes a los conjuntos que no lo son [47]. Las operaciones para operadores OR y AND son máximo y mínimo respectivamente, mientras que para la operación de complemento (NOT), NOT(A) se evaluaría como (1-A) [48].

Existen varias ventajas principales al aplicar la lógica difusa en el análisis de patrones biológicos y en las funciones de las proteínas. En primer lugar, la lógica difusa representa

intrínsecamente el ruido en los datos, ya que extrae las tendencias, no los valores precisos. Una gran cantidad de ruido es lo que se puede encontrar en la búsqueda de una función biológica, un término vagamente definido que sólo tiene sentido en un contexto. En segundo lugar, a diferencia de otros algoritmos automatizados en la toma de decisiones, como las redes neuronales, los algoritmos de lógica difusa tratan el mismo lenguaje empleado en las conversaciones del día a día, lo que hace que las predicciones de la lógica difusa sean fácilmente interpretables. En tercer lugar, no existe una fase de aprendizaje del conjunto de entrenamiento, es decir, el sistema no está obligado a intentar aprender las reglas. Las reglas llegarían a partir del conocimiento del sistema [49].

Referencias bibliográficas

[1] Martínez Ladrón de Guevara, Jorge. "BIOSEQ: una librería para bioinformática en R". Tesis T.F.M. (Trabajo Fin de Máster), Universidad Complutense de Madrid, 2013.

[2] Pelta, D.A. "Algoritmos heurísticos en bioinformática". Granada: Universidad de Granada, 2013. 182 p. [Online] Avaliable: http://hdl.handle.net/10481/24513

[3] GenBank - National Center for Biotechnology Information [Online] Avaliable: http://www.ncbi.nlm.nih.gov/genbank/

[4] UniProt (Universal Protein Resource) [Online] Avaliable: http://www.uniprot.org/

[5] Jin Xiong. "Essential Bioinformatics". Texas A&M University. Cambridge University Press, 2006. pp. 3-9.

[6] Rodrigo Santamaria. "Estadística de secuencias genómicas". Universidad de Salamanca. [Online] Avaliable: http://vis.usal.es/rodrigo/documentos/bioinformatica/temas/2_Estad%C3%ADstica%20de%20secuencias.pdf

[7] The R Project for Statistical Computing. [Online]. Avaliable: http://www.r-project.org/

[8] Kim Seefeld and Ernst Linder. "Statistics Using R with Biological Examples" Department of Mathematics & Statistics, University of New Hampshire, Durham, 2007, NH. p. 6.

[9] Bioconductor. Open Source Software for Bioinformatics. [Online]. Avaliable: http://www.bioconductor.org/

[10] About Bioconductor. [Online]. Avaliable: http://www.bioconductor.org/about/

[11] SeqinR: Biological Sequences in R project. [Online]. Avaliable: http://seqinr.r-forge.r-project.org/

[12] Tinn-R. [Online]. Avaliable: http://www.sciviews.org/Tinn-R/

[13] Revolution R. [Online]. Avaliable: http://www.revolutionanalytics.com/

[14] MinGW. Minimalist GNU for Windows. [Online]. Avaliable: http://www.mingw.org/

[15] Luis Hernández Yáñez. "Instalación de MinGW". Herramientas de desarrollo. Fundamentos de la programación. Universidad Complutense de Madrid [Online]. Avaliable: http://www.fdi.ucm.es/profesor/luis/fp/devtools/MinGW.html

[16] Datanalytics. “Rutinas de C en R”, September 2010. [Online]. Avaliable: http://www.datanalytics.com/rutinas-de-c-en-r.html

[17] Michael J. Crawley. “The R Book”, 2nd. Edition. Wiley. 2012. p. 87.

[18] Martin Morgan, Nicolas Delhomme. “R / Bioconductor for High-Throughput Sequence Analysis”. 2012. pp. 2-20.

[19] Jean-Michel Claverie and Cedric Notredame. “Bioinformatics For Dummies, 2nd Edition. Wiley Publishing, 2007. pp. 10-23.

[20] Jin Xiong. “Essential Bioinformatics”. Texas A&M University. Cambridge University Press, 2006. p. 34.

[21] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. “Algorithms” [Online]. Avaliable: http://www.cs.berkeley.edu/~vazirani/algorithms.html

[22] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. “Introduction to Algorithms” (2nd ed.), MIT Press & McGraw–Hill. 2001. pp. 327-8.

[23] Konhauser J.D.E., Velleman, D., and Wagon, S. “Which way did the Bicycle Go?” Dolciani Mathematical Expositions, No. 18. The Mathematical Association of America. 1996.

[24] Sniedovich, M. “The joy of egg-dropping in Braunschweig and Hong Kong”. Transactions on Education, 4(1). 2003. pp. 48–64.

[25] BLAST: Basic Local Alignment Search Tool. [Online]. Avaliable: http://www.ncbi.nlm.nih.gov/BLAST

[26] EMBL European Bioinformatics Institute. [Online]. Avaliable: http://www.ebi.ac.uk

[27] Jin Xiong. “Essential Bioinformatics”. Texas A&M University. Cambridge University Press, 2006. pp. 31-50.

[28] N. R. Jennings and M. Wooldridge. “Applying Agent Technology”. Journal of Applied Artificial Intelligence special issue on Intelligent Agents and Multi-Agent Systems, 1995.

[29] JADE (Java Agent Development Framework) and Open Source platform for peer-to-peer agent based applications. [Online]. Avaliable: http://jade.tilab.com/

[30] “Foundation for Intelligent Physical Agents” (FIPA). [Online]. Avaliable: http://www.fipa.org/

[31] Programación JADE. Sistemas Multiagente. Escuela Superior de Ingeniería Informática de la Universidad de Vigo. [Online]. Avaliable: http://programacionjade.wikispaces.com/

[32] Gomez, O. M., & Alvarez, P. M. "Kernel de tiempo real para control de procesos". In Conferencia de Ingeniería Eléctrica, September 2003, pp. 6-7.

[33] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco and Giovanni Rimassa. "JADE Programmer's Guide". Telecom Italia S.p.A, April 2010.

[34] RCaller. A library for calling R from Java. [Online]. Avaliable: https://code.google.com/p/rcaller/

[35] Runiversal. Paquete para convertir objetos R en variables JAVA y XML. [Online]. Avaliable: http://cran.rproject.org/web/packages/Runiversal/

[36] The Eclipse Foundation open source community website. [Online]. Avaliable: http://eclipse.org/

[37] Java Runtime Environment (JRE). [Online]. Avaliable: https://www.java.com/en/

[38] Sean R. Eddy. "What is dynamic programming?" Nature Biotechnology Vol. 22 N. 7. July 2004.

[39] Smith, Temple F. and Waterman, Michael S. "Identification of Common Molecular Subsequences". Journal of Molecular Biology 147, 1981.

[40] Patrick Aboyoun. "Pairwise Sequence Alignments". Gentleman Lab. Fred Hutchinson Cancer Research Center. Seattle, WA. August 2008. p. 23.

[41] Jorge Cordero, Jorge Martínez, Óscar Sánchez, Victoria López, Beatriz González. "BioSeq: Una libreria R para el análisis de secuencias de datos". IV Congreso Español de Informática, 2013.

[42] G-Tec. Grupo Tecnología UCM [Online]. Avaliable: http://tecnologiaucm.es/

[43] "Fuzzy Logic". Stanford Encyclopedia of Philosophy. Stanford University. 2006.

[44] Zadeh, L.A. "Fuzzy sets". Information and Control 8, 1965. pp. 338–353.

[45] Novák, V., Perfilieva, I. and Močkoř, J. "Mathematical principles of fuzzy logic". Dodrecht: Kluwer Academic, 1999.

[46] E. Cox. "Fuzzy Fundamentals", IEEE Spectrum, Volume 29, Issue 10, October 1992, pp. 58-61.

[47] "A Short Fuzzy Logic Tutorial". April 2010.

[48] Nivit Gill, Shailendra Singh. "Biological Sequence Matching Using Fuzzy Logic". International Journal of Scientific & Engineering Research Volume 2, Issue 7, July 2011.

[49] "Fuzzy Logic". Universitat Autònoma de Barcelona. p.1. [Online]. Avaliable: http://bioinf.uab.es/bypass/Additional/FuzzyLogic.pdf

Apéndice 1: Algoritmo Needleman-Wunsch modificado en R

```
NeedlemanWunsch<-function(Sec1,Sec2,Hueco,MSHeader,MSData){

 #Contador de huecos.

 ContGaps <- 0

 #Penalización por hueco (0) y matriz de sustitución por
 defecto (identidad).

 if (missing(Hueco))
 {
    Hueco <- 0
 }

 if (missing(MSData))
 {
    for (i in 1:length(MSHeader))
    {
        for (j in 1:length(MSHeader))
        {
            if (i!=j)
            {
                MSData <- c(MSData, 0)
            }else{
                if (i==1 & j ==1)
                {
                    MSData <- 1
                }else{
                    MSData <- c(MSData, 1)
                }
            }
        }
    }
 }
```

```
#Longitud fila y columna de la matriz de sustitución.

MSRowCol <- length(MSHeader)

#Definimos los valores de una matriz cuadrada que hará la
función de matriz de puntaje.

NombresF <- NombresC <- MSHeader

MatrizPuntaje <- matrix(MSData,MSRowCol,MSRowCol,byrow=TRUE,
dimnames=list(NombresC,NombresF))

#Matriz que muestra los resultados parciales de cada
posible alineamiento.

S1 <- c("",Sec1)
S2 <- c("",Sec2)

LengthS1 <- length(S1)
LengthS2 <- length(S2)

MatrizAlin <- matrix(data = NA, nrow = LengthS2, ncol =
LengthS1 , byrow = FALSE, dimnames = list(S2,S1))

for (i in 1:LengthS2){
    if(i==1)
    {
        MatrizAlin[i,1] <- 0
    }else{
        MatrizAlin[i,1] <- Hueco*i-Hueco
    }
}

for (j in 1:LengthS1){
    if(j==1)
    {
        MatrizAlin[1,j] <- 0
```

```
    }else{
        MatrizAlin[1,j] <- Hueco*j-Hueco
    }
}

for (i in 2:LengthS2){
    for (j in 2:LengthS1){
        Elem1 <- S1[j]
        Elem2 <- S2[i]

        for (k in 1:length(NombresF)){
            if(Elem1==NombresF[k]){
                PosF <- k
            }
        }

        for (l in 1:length(NombresC)){
            if(Elem2==NombresC[l]){
                PosC <- l
            }
        }

        Diag <- MatrizAlin[i-1,j-1] + MatrizPuntaje[PosF,PosC]
        Arriba <- MatrizAlin[i-1,j] + Hueco
        Izq <- MatrizAlin[i,j-1] + Hueco
        MatrizAlin[i,j] <- max(Diag,Arriba,Izq)
    }
}

#Hallamos el alineamiento óptimo de las secuencias
retrocediendo en la matriz de alineamiento.

AlineamA <- character()
AlineamB <- character()

i <- LengthS2
j <- LengthS1
```

```
while(i > 1 & j > 1){

    Punt <- MatrizAlin[i,j]
    Diag <- MatrizAlin[i-1,j-1]
    Arriba <- MatrizAlin[i-1,j]
    Izq <- MatrizAlin[i,j-1]

    Elem1 <- S1[j]
    Elem2 <- S2[i]

    for (k in 1:length(NombresF)){
        if(Elem1==NombresF[k]){
            PosF <- k
        }
    }

    for (l in 1:length(NombresC)){
        if(Elem2==NombresC[l]){
            PosC <- l
        }
    }

    if(Punt == Diag + MatrizPuntaje[PosF,PosC])
    {
        AlineamA <- c(S1[j],AlineamA)
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        j <- j-1
    }else if(Punt == Arriba + Hueco){
        AlineamA <- c("-",AlineamA)
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        ContGaps <- ContGaps + 1
    }else if(Punt == Izq + Hueco){
        AlineamA <- c(S1[j],AlineamA)
        AlineamB <- c("-",AlineamB)
        j <- j-1
```

```
        ContGaps <- ContGaps + 1
    }
}

while(i > 1)
{
    AlineamA <- c("-",AlineamA)
    AlineamB <- c(S2[i],AlineamB)
    i <- i-1
    ContGaps <- ContGaps + 1
}

while(j > 1)
{
    AlineamA <- c(S1[j],AlineamA)
    AlineamB <- c("-",AlineamB)
    j <- j-1
    ContGaps <- ContGaps + 1
}

#Método gráfico por puntos para identificar las
regiones con mayor similitud entre las dos secuencias.

library("seqinr")
dotPlot(Sec1,Sec2)

#Mostramos las secuencias a alinear.

print("Sequence 1")
cat(Sec1, "\n\n")
print("Sequence 2")
cat(Sec2, "\n\n")

#Mostramos la matriz de sustitución y de alineamientos
parciales.
```

```
 print("Substitution Matrix")
 print(MatrizPuntaje)
 cat("\n")
 print("Alignment Matrix")
 print(MatrizAlin)
 cat("\n")

 #Mostramos el alineamiento óptimo de las secuencias.

 print ("Sequence 1 Optimal Alignment")
 cat(AlineamA, "\n\n")
 print ("Sequence 2 Optimal Alignment")
 cat(AlineamB, "\n\n")

 #Mostramos la puntuación del alineamiento óptimo.

 print("Optimal Alignment Score")
 cat(MatrizAlin[LengthS2,LengthS1], "\n\n")

 #Mostramos el número de huecos utilizados para el
 alineamiento.

 print("Number of Gaps")
 cat(ContGaps, "\n\n")
}
```

Apéndice 2: Algoritmo Needleman-Wunsch modificado en R con librerías dinámicas

```
NeedlemanWunsch<-function(Sec1,Sec2,Hueco,MSHeader,MSData){

    #Contador de huecos.

    ContGaps <- 0

    #Penalización por hueco (0) y matriz de sustitución por defecto
    (identidad).

    if (missing(Hueco))
    {
        Hueco <- 0
    }

    if (missing(MSData))
    {
        for (i in 1:length(MSHeader))
        {
            for (j in 1:length(MSHeader))
            {
                if (i!=j)
                {
                    MSData <- c(MSData, 0)
                }else{
                    if (i==1 & j ==1)
                    {
                        MSData <- 1
                    }else{
                        MSData <- c(MSData, 1)
                    }
                }
            }
        }
```

```
}

#Longitud fila y columna de la matriz de sustitución.

MSRowCol <- length(MSHeader)

#Definimos los valores de una matriz cuadrada que hará la
   función de   matriz de puntaje.

NombresF <- NombresC <- MSHeader

MatrizPuntaje <- matrix(MSData,MSRowCol,MSRowCol,byrow=TRUE,
dimnames=list(NombresC,NombresF))

#Matriz que muestra los resultados parciales de cada posible
alineamiento.

S1 <- c("",Sec1)
S2 <- c("",Sec2)

LengthS1 <- length(S1)
LengthS2 <- length(S2)
LengthMSHeader <- length(MSHeader)
LengthMSData <- length(MSData)

MatrizAlin <- matrix(data = 0, nrow = LengthS2, ncol = LengthS1)

dyn.load("C:/Users/oscarsb1988/Dropbox/Educación/UCM/
Proyecto Final de Master/Proyecto/Compilador Externo/NW.dll")

#Asociación numérica a caracteres para comparación/asignación en
DLL.

MSHeaderN <- c(0)
j <- 1
```

```
for(i in 1:length(MSHeader))
{
   MSHeaderN[j] <- i
   j <- j+1
}

S1N <- c(0)
S1N[1] <- 0

for(i in 1:length(S1))
{
   for(j in 1:length(MSHeader))
   {
      if(S1[i]==MSHeader[j])
      {
         S1N[i] <- j
      }
   }
}

S2N <- c(0)
S2N[1] <- 0

for(i in 1:length(S2))
{
   for(j in 1:length(MSHeader))
   {
      if(S2[i]==MSHeader[j])
      {
         S2N[i] <- j
      }
   }
}

#Llamada a la función .C para comenzar con el cómputo en DLL.

MatrizAlin <-.C("NW",resultado=as.vector(as.integer(MatrizAlin)),
```

```
as.vector(as.integer(MatrizPuntaje)),as.vector(as.integer(S1N)),
as.vector(as.integer(S2N)),as.integer(Hueco),
as.vector(as.integer(MSHeaderN)),as.integer(LengthS1),
as.integer(LengthS2),as.integer(LengthMSHeader),
as.integer(LengthMSData)) $resultado

MatrizAlin <- matrix(data = MatrizAlin, nrow = LengthS2, ncol =
LengthS1 , byrow = TRUE, dimnames = list(S2,S1))

dyn.unload("C:/Users/oscarsb1988/Dropbox/Educación/UCM/
Proyecto Final de Master/Proyecto/Compilador Externo/NW.dll")

#Hallamos el alineamiento óptimo de las secuencias retrocediendo
en la matriz de alineamiento.

AlineamA <- character()
AlineamB <- character()

i <- LengthS2
j <- LengthS1

while(i > 1 & j > 1){

    Punt <- MatrizAlin[i,j]
    Diag <- MatrizAlin[i-1,j-1]
    Arriba <- MatrizAlin[i-1,j]
    Izq <- MatrizAlin[i,j-1]

    Elem1 <- S1[j]
    Elem2 <- S2[i]

    for (k in 1:length(NombresF)){
        if(Elem1==NombresF[k]){
            PosF <- k
        }
    }
```

```
    for (l in 1:length(NombresC)){
        if(Elem2==NombresC[l]){
            PosC <- l
        }
    }

    if(Punt == Diag + MatrizPuntaje[PosF,PosC])
    {
        AlineamA <- c(S1[j],AlineamA)
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        j <- j-1
    }else if(Punt == Arriba + Hueco){
        AlineamA <- c("-",AlineamA)
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        ContGaps <- ContGaps + 1
    }else if(Punt == Izq + Hueco){
        AlineamA <- c(S1[j],AlineamA)
        AlineamB <- c("-",AlineamB)
        j <- j-1
        ContGaps <- ContGaps + 1
    }
}

while(i > 1)
{
    AlineamA <- c("-",AlineamA)
    AlineamB <- c(S2[i],AlineamB)
    i <- i-1
    ContGaps <- ContGaps + 1
}

while(j > 1)
{
    AlineamA <- c(S1[j],AlineamA)
    AlineamB <- c("-",AlineamB)
```

```
    j <- j-1
    ContGaps <- ContGaps + 1
}

#Método gráfico por puntos para identificar las regiones con mayor
similitud entre las dos secuencias.

library("seqinr")
dotPlot(Sec1,Sec2)

#Mostramos las secuencias a alinear.

print("Sequence 1")
cat(Sec1, "\n\n")
print("Sequence 2")
cat(Sec2, "\n\n")

#Mostramos la matriz de sustitución y de alineamientos parciales.

print("Substitution Matrix")
print(MatrizPuntaje)
cat("\n")
print("Alignment Matrix")
print(MatrizAlin)
cat("\n")

#Mostramos el alineamiento óptimo de las secuencias.

print ("Sequence 1 Optimal Alignment")
cat(AlineamA, "\n\n")
print ("Sequence 2 Optimal Alignment")
cat(AlineamB, "\n\n")

#Mostramos la puntuación del alineamiento óptimo.

print("Optimal Alignment Score")
cat(MatrizAlin[LengthS2,LengthS1], "\n\n")
```

```
    #Mostramos el número de huecos utilizados para el alineamiento.

    print("Number of Gaps")
    cat(ContGaps, "\n\n")
}

void NW(int *MAlin, int *MPuntaje, int *Sec1, int *Sec2, int *Gap, int
*MPHeader, int *LengthS1, int *LengthS2, int *LengthMSHeader, int
*LengthMSData){

    int h,i,j,k,l;
    int ma[*LengthS2][*LengthS1];
    int mp[*LengthMSHeader][*LengthMSHeader];
    int PosF,PosC,Diag,Arriba,Izq;
    int May,Elem1,Elem2;

    //Convertimos el vector "MatrizAlin" en matriz.

    h=0;
    for(i=0;i<*LengthS2;i++){
        for(j=0;j<*LengthS1;j++){
            ma[i][j]=MAlin[h];
            h++;
        }
    }

    //Convertimos el vector "MPuntaje" en matriz.

    h=0;
    for(i=0;i<*LengthMSHeader;i++){
        for(j=0;j<*LengthMSHeader;j++){
            mp[i][j]=MPuntaje[h];
            h++;
        }
    }
```

```
//Matriz que muestra los resultados parciales de cada
posible alineamiento.

for (i=0;i<*LengthS2;i++)
{
       ma[i][0] = (*Gap)*i;
}

for (j=0;j<*LengthS1;j++)
{
       ma[0][j] = (*Gap)*j;
}

for (i=1;i<*LengthS2;i++){
    for (j=1;j<*LengthS1;j++){

       Elem1 = Sec1[j];
       Elem2 = Sec2[i];

       for (k=0;k<*LengthMSHeader;k++){
          if(Elem1==MPHeader[k])
          {
             PosF = k;
          }
       }

       for (l=0;l<*LengthMSHeader;l++){
          if(Elem2==MPHeader[l])
          {
             PosC = l;
          }
       }

       Diag = ma[i-1][j-1] + mp[PosF][PosC];
       Arriba = ma[i-1][j] + (*Gap);
       Izq = ma[i][j-1] + (*Gap);
       //ma[i][j] = max(Diag,Arriba,Izq);
```

```
            May = Diag;

            if (Arriba > May)
            {
               May = Arriba;
            }
            if (Izq > May)
            {
               May = Izq;
            }

            ma[i][j] = May;
         }
      }

      //Conversión de matriz de alineamiento(ma) en vector.

      h=0;
      for(i=0;i<*LengthS2;i++){
         for(j=0;j<*LengthS1;j++){
            MAlin[h]=ma[i][j];
            h++;
         }
      }
}
```

Apéndice 3: Algoritmo Smith-Waterman modificado en R

```
SmithWaterman <- function(Sec1,Sec2,Hueco,MSHeader,MSData){

 #Contador de huecos.

 ContGaps <- 0

 #Longitud fila y columna de la matriz de sustitución.

 MSRowCol <- length(MSHeader)

 #Definimos los valores de una matriz cuadrada que hará
 la función de matriz de puntaje.

 NombresF <- NombresC <- MSHeader

 MatrizPuntaje <- matrix (MSData, MSRowCol, MSRowCol,
 byrow=TRUE, dimnames=list(NombresC,NombresF))

 #Convertimos los valores negativos de la matriz de
    puntaje a 0.

 for (i in 1:length(NombresF)){
    for (j in 1:length(NombresC)){
       if(MatrizPuntaje[i,j] < 0){
          MatrizPuntaje[i,j] <- 0
       }
    }
 }

 #Matriz que muestra los resultados parciales de cada
 posible alineamiento.

 S1 <- c("",Sec1)
 S2 <- c("",Sec2)
```

```
LengthS1 <- length(S1)
LengthS2 <- length(S2)
MatrizAlin <- matrix(data = NA, nrow = LengthS2, ncol =
LengthS1 , byrow = FALSE, dimnames = list(S2,S1))

for (i in 1:LengthS2){
   if(i==1)
   {
      MatrizAlin[i,1] <- 0
   }else{
      MatrizAlin[i,1] <- Hueco*i-Hueco
      if(MatrizAlin[i,1] < 0)
      {
         MatrizAlin[i,1] <- 0
      }
   }
}

for (j in 1:LengthS1){
   if(j==1)
   {
      MatrizAlin[1,j] <- 0
   }else{
      MatrizAlin[1,j] <- Hueco*j-Hueco
      if(MatrizAlin[1,j] < 0)
      {
         MatrizAlin[1,j] <- 0
      }
   }
}

for (i in 2:LengthS2){
   for (j in 2:LengthS1){
      Elem1 <- S1[j]
      Elem2 <- S2[i]
```

```
        for (k in 1:length(NombresF)){
            if(Elem1==NombresF[k]){
                PosF <- k
            }
        }

        for (l in 1:length(NombresC)){
            if(Elem2==NombresC[l]){
                PosC <- l
            }
        }

        Diag <- MatrizAlin[i-1,j-1] +
        MatrizPuntaje[PosF,PosC]
        Arriba <- MatrizAlin[i-1,j] + Hueco
        Izq <- MatrizAlin[i,j-1] + Hueco
        MatrizAlin[i,j] <- max(Diag,Arriba,Izq)
    }
}

#Hallamos el alineamiento óptimo de las secuencias
retrocediendo en la matriz de alineamiento.

AlineamA <- character()
AlineamB <- character()

MaxVal <- 0

for (i in 2:LengthS2){
    for (j in 2:LengthS1){
        if(MaxVal < MatrizAlin[i,j]){
            MaxVal <- MatrizAlin[i,j]
            MaxVali <- i
            MaxValj <- j
        }
    }
}
```

```
i <- MaxVali
j <- MaxValj

while((i > 1 & j > 1) | (MatrizAlin[i,j] > 0)){
    Punt <- MatrizAlin[i,j]
    Diag <- MatrizAlin[i-1,j-1]
    Arriba <- MatrizAlin[i-1,j]
    Izq <- MatrizAlin[i,j-1]

    Elem1 <- S1[j]
    Elem2 <- S2[i]

    for (k in 1:length(NombresF)){
        if(Elem1==NombresF[k]){
            PosF <- k
        }
    }

    for (l in 1:length(NombresC)){
        if(Elem2==NombresC[l]){
        PosC <- l
        }
    }

    if(Punt == Diag + MatrizPuntaje[PosF,PosC])
    {
        AlineamA <- c(S1[j],AlineamA)
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        j <- j-1
    }else if(Punt == Arriba + Hueco){
        AlineamA <- c("-",AlineamA)
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        ContGaps <- ContGaps + 1
    }else if(Punt == Izq + Hueco){
        AlineamA <- c(S1[j],AlineamA)
```

```
        AlineamB <- c("-",AlineamB)
        j <- j-1
        ContGaps <- ContGaps + 1
    }
}

while((i > 1) & (MatrizAlin[i,j] > 0))
{
    AlineamA <- c("-",AlineamA)
    AlineamB <- c(S2[i],AlineamB)
    i <- i-1
    ContGaps <- ContGaps + 1
}

while((j > 1) & (MatrizAlin[i,j] > 0))
{
    AlineamA <- c(S1[j],AlineamA)
    AlineamB <- c("-",AlineamB)
    j <- j-1
    ContGaps <- ContGaps + 1
}

#Método gráfico por puntos para identificar las
regiones con mayor similitud entre las dos secuencias.

library ("seqinr")
dotPlot(Sec1,Sec2)

#Mostramos las secuencias a alinear.

print("Sequence 1")
cat(Sec1, "\n\n")
print("Sequence 2")
cat(Sec2, "\n\n")

#Mostramos la matriz de sustitución y de alineamientos
parciales.
```

```
print("Substitution Matrix")
print(MatrizPuntaje)

cat("\n")
print("Alignment Matrix")
print(MatrizAlin)
cat("\n")

#Mostramos el alineamiento óptimo de las secuencias.

print ("Sequence 1 Optimal Alignment")
cat(AlineamA, "\n\n")
print ("Sequence 2 Optimal Alignment")
cat(AlineamB, "\n\n")

#Mostramos la puntuación del alineamiento óptimo.

print("Optimal Alignment Score")
cat(MatrizAlin[MaxVali,MaxValj], "\n\n")

#Mostramos el número de huecos utilizados para el
alineamiento.

print("Number of Gaps")
cat(ContGaps, "\n\n")
}
```

Apéndice 4: Algoritmo Smith-Waterman modificado en R con librerías dinámicas

```
SmithWaterman <- function(Sec1,Sec2,Hueco,MSHeader,MSData){

    #Contador de huecos.

    ContGaps <- 0

    #Penalización por hueco (0) y matriz de sustitución por
    defecto (identidad).

    if (missing(Hueco))
    {
        Hueco <- 0
    }

    if (missing(MSData))
    {
        for (i in 1:length(MSHeader))
        {
            for (j in 1:length(MSHeader))
            {
                if (i!=j)
                {
                    MSData <- c(MSData, 0)
                }else{
                    if (i==1 & j ==1)
                    {
                        MSData <- 1
                    }else{
                        MSData <- c(MSData, 1)
                    }
                }
            }
```

```
    }
}

#Longitud fila y columna de la matriz de sustitución.

MSRowCol <- length(MSHeader)

#Definimos los valores de una matriz cuadrada que hará la
función de matriz de puntaje.

NombresF <- NombresC <- MSHeader

MatrizPuntaje <- matrix(MSData,MSRowCol,MSRowCol,byrow=TRUE,
dimnames=list(NombresC,NombresF))

#Convertimos los valores negativos de la matriz de puntaje a 0.

for (i in 1:length(NombresF)){
    for (j in 1:length(NombresC)){
        if(MatrizPuntaje[i,j] < 0){
            MatrizPuntaje[i,j] <- 0
        }
    }
}

#Matriz que muestra los resultados parciales de cada posible
alineamiento.

S1 <- c("",Sec1)
S2 <- c("",Sec2)

LengthS1 <- length(S1)
LengthS2 <- length(S2)
LengthMSHeader <- length(MSHeader)
LengthMSData <- length(MSData)

MatrizAlin <- matrix(data = 0, nrow = LengthS2, ncol = LengthS1)
```

```
dyn.load("C:/Users/oscarsb1988/Dropbox/Educación/UCM/
Proyecto Final de Master/Proyecto/Compilador Externo/
SW.dll")

#Asociación numérica a caracteres para comparación/asignación
en DLL.

MSHeaderN <- c(0)
j <- 1

for(i in 1:length(MSHeader))
{
   MSHeaderN[j] <- i
   j <- j+1
}

S1N <- c(0)
S1N[1] <- 0

for(i in 1:length(S1))
{
   for(j in 1:length(MSHeader))
   {
      if(S1[i]==MSHeader[j])
      {
         S1N[i] <- j
      }
   }
}

S2N <- c(0)
S2N[1] <- 0
```

```
for(i in 1:length(S2))
{
    for(j in 1:length(MSHeader))
    {
        if(S2[i]==MSHeader[j])
        {
            S2N[i] <- j
        }
    }
}

#Llamada a la función .C para comenzar con el cómputo en DLL

MatrizAlin <- .C("SW",
resultado=as.vector(as.integer(MatrizAlin)),
as.vector(as.integer(MatrizPuntaje)),
as.vector(as.integer(S1N)), as.vector(as.integer(S2N)),
as.integer(Hueco), as.vector(as.integer(MSHeaderN)),
as.integer(LengthS1), as.integer(LengthS2),
as.integer(LengthMSHeader), as.integer(LengthMSData))$resultado

MatrizAlin <- matrix(data = MatrizAlin, nrow = LengthS2, ncol =
LengthS1 , byrow = TRUE, dimnames = list(S2,S1))

dyn.unload("C:/Users/oscarsb1988/Dropbox/Educación/UCM/
Proyecto Final de Master/Proyecto/Compilador Externo/SW.dll")

#Hallamos el alineamiento óptimo de las secuencias
retrocediendo en la matriz de alineamiento.

AlineamA <- character()
AlineamB <- character()

MaxVal <- 0
```

```
for (i in 2:LengthS2){
    for (j in 2:LengthS1){
        if(MaxVal < MatrizAlin[i,j]){
            MaxVal <- MatrizAlin[i,j]
            MaxVali <- i
            MaxValj <- j
        }
    }
}

i <- MaxVali
j <- MaxValj

while((i > 1 & j > 1) | (MatrizAlin[i,j] > 0)){
    Punt <- MatrizAlin[i,j]
    Diag <- MatrizAlin[i-1,j-1]
    Arriba <- MatrizAlin[i-1,j]
    Izq <- MatrizAlin[i,j-1]

    Elem1 <- S1[j]
    Elem2 <- S2[i]

    for (k in 1:length(NombresF)){
        if(Elem1==NombresF[k]){
            PosF <- k
        }
    }

    for (l in 1:length(NombresC)){
        if(Elem2==NombresC[l]){
        PosC <- l
        }
    }

    if(Punt == Diag + MatrizPuntaje[PosF,PosC])
    {
        AlineamA <- c(S1[j],AlineamA)
```

```
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        j <- j-1
    }else if(Punt == Arriba + Hueco){
        AlineamA <- c("-",AlineamA)
        AlineamB <- c(S2[i],AlineamB)
        i <- i-1
        ContGaps <- ContGaps + 1
    }else if(Punt == Izq + Hueco){
        AlineamA <- c(S1[j],AlineamA)
        AlineamB <- c("-",AlineamB)
        j <- j-1
        ContGaps <- ContGaps + 1
    }
}

while((i > 1) & (MatrizAlin[i,j] > 0))
{
    AlineamA <- c("-",AlineamA)
    AlineamB <- c(S2[i],AlineamB)
    i <- i-1
    ContGaps <- ContGaps + 1
}

while((j > 1) & (MatrizAlin[i,j] > 0))
{
    AlineamA <- c(S1[j],AlineamA)
    AlineamB <- c("-",AlineamB)
    j <- j-1
    ContGaps <- ContGaps + 1
}

#Método gráfico por puntos para identificar las regiones con
mayor similitud entre las dos secuencias.

library ("seqinr")
dotPlot(Sec1,Sec2)
```

```
    #Mostramos las secuencias a alinear.

    print("Sequence 1")
    cat(Sec1, "\n\n")
    print("Sequence 2")
    cat(Sec2, "\n\n")

    #Mostramos la matriz de sustitución y de alineamientos parciales.

    print("Substitution Matrix")
    print(MatrizPuntaje)
    cat("\n")
    print("Alignment Matrix")
    print(MatrizAlin)
    cat("\n")

    #Mostramos el alineamiento óptimo de las secuencias.

    print ("Sequence 1 Optimal Alignment")
    cat(AlineamA, "\n\n")
    print ("Sequence 2 Optimal Alignment")
    cat(AlineamB, "\n\n")

    #Mostramos la puntuación del alineamiento óptimo.

    print("Optimal Alignment Score")
    cat(MatrizAlin[MaxVali,MaxValj], "\n\n")

    #Mostramos el número de huecos utilizados para el
    alineamiento.

    print("Number of Gaps")
    cat(ContGaps, "\n\n")
}
```

```
void SW(int *MAlin, int *MPuntaje, int *Sec1, int *Sec2, int *Gap, int
*MPHeader, int *LengthS1, int *LengthS2, int *LengthMSHeader, int
*LengthMSData){

    int h,i,j,k,l;
    int ma[*LengthS2][*LengthS1];

    int mp[*LengthMSHeader][*LengthMSHeader];
    int PosF,PosC,Diag,Arriba,Izq;
    int May,Elem1,Elem2;

    //Convertimos el vector "MatrizAlin" en matriz.

    h=0;
    for(i=0;i<*LengthS2;i++){
        for(j=0;j<*LengthS1;j++){
            ma[i][j]=MAlin[h];
            h++;
        }
    }

    //Convertimos el vector "MPuntaje" en matriz.

    h=0;
    for(i=0;i<*LengthMSHeader;i++){
        for(j=0;j<*LengthMSHeader;j++){
            mp[i][j]=MPuntaje[h];
            h++;
        }
    }

    //Matriz que muestra los resultados parciales de cada
    posible alineamiento.

    for (i=0;i<*LengthS2;i++)
    {
        ma[i][0] = (*Gap)*i;
```

```
    if(ma[i][0] < 0)
    {
        ma[i][0] = 0;
    }
}

for (j=0;j<*LengthS1;j++)
{
    ma[0][j] = (*Gap)*j;
    if(ma[0][j] < 0)
    {
        ma[0][j] = 0;
    }
}

for (i=1;i<*LengthS2;i++){
    for (j=1;j<*LengthS1;j++){

    Elem1 = Sec1[j];
    Elem2 = Sec2[i];

    for (k=0;k<*LengthMSHeader;k++){
        if(Elem1==MPHeader[k])
        {
            PosF = k;
        }
    }

    for (l=0;l<*LengthMSHeader;l++){
        if(Elem2==MPHeader[l])
        {
            PosC = l;
        }
    }
```

```
            Diag = ma[i-1][j-1] + mp[PosF][PosC];
            Arriba = ma[i-1][j] + (*Gap);
            Izq = ma[i][j-1] + (*Gap);
            //ma[i][j] = max(Diag,Arriba,Izq);

            May = Diag;

            if (Arriba > May)
            {
                May = Arriba;
            }
            if (Izq > May)
            {
                May = Izq;
            }

            ma[i][j] = May;

            }
        }

        //Conversión de matriz de alineamiento(ma) en vector.

        h=0;
        for(i=0;i<*LengthS2;i++){
            for(j=0;j<*LengthS1;j++){
                MAlin[h]=ma[i][j];
                h++;
            }
        }
}
```