UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Desarrollo de un algoritmo paralelo para la búsqueda de patrones proteína-ligando

LUCKAS STRNAD

Profesor Guía: RENZO ANGLES

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
diciembre, 2025

UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

Desarrollo de un algoritmo paralelo para la búsqueda de patrones proteína-ligando

LUCKAS STRNAD

Profesor Guía: RENZO ANGLES

Profesor Informante: INFORMANTE 1 (PENDIENTE)

Profesor Informante: INFORMANTE 2 (PENDIENTE)

Memoria para optar al título de
Ingeniero Civil en Computación

El presente documento fue calificado con nota: _____

Curicó – Chile

diciembre, 2025

Dedicado a ...

AGRADECIMIENTOS

Agradecimientos a ...

TABLA DE CONTENIDOS

página

Anexos

ÍNDICE DE FIGURAS

página

ÍNDICE DE TABLAS

página

RESUMEN

La identificación de patrones en proteínas es un desafío clave en bioinformática, esencial para comprender cómo interactúan los aminoácidos con otras moléculas como los ligandos. Las aproximaciones tradicionales que generan todas las subsecuencias posibles sufren un crecimiento combinatorio que limita su uso práctico debido al elevado coste computacional.

Para abordar esta problemática se desarrolló un algoritmo paralelo basado en la estrategia de la *Longest Common Subsequence* (LCS). La solución reemplaza la enumeración exhaustiva de subsecuencias por una formulación de programación dinámica que calcula las coincidencias sobre una matriz $n \times m$ y permite recuperar alineamientos compatibles con la notación Prosite. La implementación, construida en Golang y apoyada en Biopython para el preprocesamiento, distribuye el cálculo y la reconstrucción de la LCS entre múltiples hilos para reducir los tiempos de búsqueda frente al enfoque secuencial.

Si bien el algoritmo LCS-paralelo mitiga la explosión combinatoria del método previo, su eficiencia sigue dependiendo de la longitud de las secuencias, del número de alineamientos necesarios y de las características del hardware disponible. Futuras optimizaciones pueden considerar técnicas avanzadas de poda y heurísticas de alineamiento para escalar a volúmenes aún mayores sin perder precisión biológica.

La proyección de este trabajo es promisoria en contextos como el diseño de fármacos y la ingeniería de proteínas, donde se requiere identificar patrones proteína-ligando de manera rápida y confiable. También se contempla la integración con otras herramientas bioinformáticas para habilitar análisis complejos en tiempo real y extender la aplicabilidad del algoritmo a estudios de mayor envergadura.

1. Introducción

La tarea de descubrir patrones en proteínas es fundamental en bioinformática y biología computacional. Los algoritmos que permiten identificar secuencias de aminoácidos relevantes son esenciales para comprender la estructura y función de las proteínas, así como sus interacciones con otras moléculas, como los ligandos. Sin embargo, estos algoritmos suelen enfrentar desafíos significativos en términos de eficiencia y tiempo de procesamiento.

1.1 Contexto

Los algoritmos para comparar secuencias de aminoácidos suelen apoyarse en técnicas de programación dinámica como la *Longest Common Subsequence* (LCS) para identificar alineamientos significativos. Este enfoque permite capturar coincidencias relevantes sin necesidad de explorar cada subsecuencia de forma explícita, lo que lo convierte en una herramienta ampliamente utilizada en bioinformática para analizar similitudes estructurales y funcionales entre proteínas.

1.2 Motivación y problemática

La versión inicial de este proyecto recurría a la generación exhaustiva de subsecuencias y a su comparación posterior, lo que producía un crecimiento combinatorio que limitaba la escalabilidad del método. La necesidad de un algoritmo que mantenga la calidad de los patrones pero reduzca drásticamente el número de comparaciones motivó la adopción del enfoque LCS. Replantear el problema en términos de programación dinámica permite acotar la complejidad a $O(nm)$, mejorar el aprovechamien-

to de la computación paralela y habilitar análisis oportunos en estudios con grandes volúmenes de datos.

1.3 Objetivos

Objetivo general

- Desarrollar una versión paralelizada de un algoritmo para el descubrimiento de patrones estructurales en proteínas

Objetivos específicos

- Identificar las partes del algoritmo que pueden ser paralelizadas.
- Diseñar un algoritmo que haga uso de hilos para descubrir patrones usando computación paralela.
- Implementar el algoritmo basado en hilos.
- Evaluar el rendimiento del algoritmo en paralelo frente a la versión secuencial.

1.4 Metodología

El desarrollo del proyecto seguirá un enfoque estructurado que combina el método científico y la evaluación cuantitativa para garantizar un proceso riguroso y basado en evidencia. A continuación, se describen las etapas de cada metodología aplicada en el proyecto.

1.4.1 Etapas del Método Científico

El proyecto se desarrollará siguiendo las etapas clásicas del método científico, asegurando un enfoque sistemático y objetivo:

Observación Se inicia con la revisión y análisis del algoritmo secuencial existente, identificando el problema central: la explosión combinatoria que limita su eficiencia. Esta etapa implica recopilar datos y entender a fondo el comportamiento actual del algoritmo.

Formulación de hipótesis Basándose en la observación, se planteará la hipótesis de que la paralelización mediante hilos mejorará significativamente el tiempo de ejecución del algoritmo sin comprometer la calidad de los resultados.

Experimentación Se implementará la versión paralelizada del algoritmo y se llevarán a cabo experimentos para poner a prueba la hipótesis. Esta etapa incluirá la ejecución del algoritmo bajo diferentes configuraciones, variando el número de hilos y analizando cómo se distribuye la carga de trabajo.

Análisis de resultados Los datos obtenidos durante la experimentación se analizarán para verificar si la paralelización cumple con las expectativas de mejora de rendimiento. Se compararán los tiempos de ejecución y el uso de recursos entre la versión secuencial y la paralela.

Conclusiones Basándose en el análisis de los resultados, se extraerán conclusiones sobre la efectividad de la paralelización. Si los resultados son positivos, se documentará el éxito del enfoque; en caso contrario, se propondrán nuevas iteraciones o ajustes.

1.4.2 Etapas de la Evaluación Cuantitativa

La evaluación cuantitativa complementa el uso del método científico mediante la recolección y análisis de datos numéricos para medir y validar los resultados:

Definición de métricas Se identificarán las métricas clave, como el tiempo de ejecución y el uso de recursos.

Recolección de datos Durante la etapa de experimentación, se registrarán los tiempos de ejecución y el uso de recursos del sistema en múltiples escenarios de prueba.

Análisis estadístico Los datos recolectados serán analizados estadísticamente para identificar patrones, diferencias significativas entre la versión secuencial y paralelizada, y la eficiencia de la paralelización. Este análisis permitirá validar si la

paralelización logra mejorar el rendimiento del algoritmo conforme a los objetivos del proyecto.

Interpretación y presentación Los resultados del análisis cuantitativo se interpretarán y se incluirán en la documentación final del proyecto, mostrando de forma clara el impacto de la paralelización en el rendimiento del algoritmo.

Este enfoque metodológico, que combina el Método Científico con una evaluación cuantitativa detallada, asegura que el proyecto se desarrolle de manera sistemática y objetiva, permitiendo validar de forma rigurosa las mejoras obtenidas y la consecución de los objetivos propuestos.

1.5 Plan de Trabajo

El plan de trabajo del proyecto se estructura en varias etapas que cubren desde el análisis preliminar hasta la documentación y evaluación final. Cada etapa está diseñada para asegurar un desarrollo metódico, con actividades claramente definidas y recursos adecuados para alcanzar los objetivos planteados.

1.5.1 Etapa 1: Análisis y Estudio Inicial

En esta primera etapa, se realizará un análisis exhaustivo del algoritmo secuencial. El objetivo es entender en profundidad su funcionamiento y detectar las partes críticas que pueden ser paralelizadas para mejorar su eficiencia.

1.5.2 Etapa 2: Diseño del Algoritmo Paralelo

El diseño del algoritmo paralelo se basará en los resultados obtenidos en la etapa anterior. En esta fase, se planificará cómo distribuir las tareas entre múltiples hilos de forma eficiente, asegurando un balance óptimo de carga. La estrategia de paralelización será desarrollada considerando las mejores prácticas en el uso de hilos en Golang. Se utilizarán referencias de documentación técnica y ejemplos previos de paralelización como base para estructurar un diseño que permita mejorar el rendimiento sin comprometer la precisión de los resultados.

1.5.3 Etapa 3: Implementación y Desarrollo

La implementación del algoritmo paralelo es una de las etapas más críticas del proyecto. Se llevará a cabo en Golang, aprovechando la capacidad de procesamiento multihilo de las herramientas de desarrollo disponibles. Durante esta fase, se trabajará en asegurar la correcta gestión de la concurrencia y en optimizar el código para maximizar la eficiencia de los hilos. Se utilizará un control de versiones como Git para mantener un seguimiento riguroso del progreso y garantizar la integridad del desarrollo.

1.5.4 Etapa 4: Pruebas y Validación

La etapa de pruebas y validación es esencial para comprobar la efectividad del algoritmo paralelo. Se realizarán pruebas para intentar asegurar la integridad y coherencia de los datos generados. Además, se comparará el rendimiento de la versión paralela con la versión secuencial para cuantificar las mejoras en tiempo de ejecución. Los datos de prueba se obtendrán de bases de datos de proteínas de acceso público, y se utilizarán herramientas de monitoreo de rendimiento para un análisis detallado.

1.5.5 Etapa 5: Documentación de la Memoria

La documentación del proyecto se realizará de manera meticulosa, incluyendo todos los aspectos del desarrollo, desde los antecedentes y la metodología empleada hasta los resultados obtenidos y las conclusiones. Esta etapa implicará el uso de herramientas de procesamiento de texto como LaTeX para asegurar una presentación profesional y ordenada del contenido. La memoria será un reflejo integral del trabajo realizado, proporcionando una base sólida para su evaluación académica.

1.6 Resultados Esperados

Objetivos específicos

Objetivo Específico 1: Identificar las partes del algoritmo que pueden ser paralelizadas.

- **Resultado Esperado:** Se espera identificar de manera clara las secciones del algoritmo secuencial que presentan alta carga computacional y que pueden beneficiarse de la paralelización.
- **Pruebas para Lograr Resultado Esperado:** Identificación de cuellos de botella a través de pruebas de rendimiento.

Objetivo Específico 2: Diseñar un algoritmo que haga uso de hilos para descubrir patrones usando computación paralela.

- **Resultado Esperado:** Se espera tener un diseño detallado y optimizado del algoritmo que divida el trabajo en tareas paralelas de forma efectiva, manteniendo la precisión del análisis.
- **Pruebas para Lograr Resultado Esperado:** Validación del diseño mediante revisiones por pares y simulaciones de ejecución en entornos controlados para evaluar la distribución de carga y sincronización de los hilos.

Objetivo Específico 3: Implementar el algoritmo basado en hilos.

- **Resultado Esperado:** Se espera que el algoritmo sea implementado con éxito y que ejecute de forma paralela el cálculo de las matrices LCS y la reconstrucción de patrones sin errores de concurrencia.
- **Pruebas para Lograr Resultado Esperado:** Ejecuciones de prueba del código en entornos de desarrollo, uso de pruebas unitarias para validar la funcionalidad de cada parte del código, y monitoreo del uso de CPU y memoria durante la ejecución.

Objetivo Específico 4: Evaluar el rendimiento del algoritmo en paralelo frente a la versión en serie.

- **Resultado Esperado:** Se espera que el algoritmo en paralelo reduzca significativamente el tiempo de procesamiento en al menos un 50% respecto a la versión en serie, sin comprometer la precisión de los resultados.

- **Pruebas para Lograr Resultado Esperado:** Comparación de tiempos de ejecución entre la versión paralela y la versión secuencial del algoritmo en diferentes conjuntos de datos de prueba, y análisis de la eficiencia de los hilos en función del número de hilos usados.

1.7 Solución

1.8 Contribuciones

2. Contexto

2.1 Proteínas, ligandos y cadenas

Las definiciones formales de algunos conceptos importantes son:

- Aminoácidos: Son moléculas que se combinan para formar proteínas.[4]
- Proteínas: Son moléculas grandes y complejas que desempeñan muchas funciones críticas en el cuerpo. Realizan la mayor parte del trabajo en las células y son necesarias para la estructura, función y regulación de los tejidos y órganos del cuerpo.[5]
- Ligandos: Es una sustancia (usualmente una molécula pequeña) que forma un complejo con una biomolécula. En un sentido más estricto, es una molécula que envía una señal al unirse al centro activo de una proteína. [6] Es decir, un ligando es una molécula que se une a una proteína, alterando su estructura y función.

Hay 20 tipos diferentes de aminoácidos que se pueden combinar para formar una proteína. La secuencia de aminoácidos determina la estructura tridimensional única de cada proteína y su función específica [5].

El Cuadro 2.1 contiene la lista de aminoácidos estándar.

Nombre	Abreviatura	Símbolo
Alanina	Ala	A
Arginina	Arg	R
Asparagina	Asn	N
Aspártico	Asp	D
Cisteína	Cys	C
Fenilalanina	Phe	F
Glicina	Gly	G
Glutámico	Glu	E
Glutamina	Gln	Q
Histidina	His	H
Isoleucina	Ile	I
Leucina	Leu	L
Lisina	Lys	K
Metionina	Met	M
Prolina	Pro	P
Serina	Ser	S
Tirosina	Tyr	Y
Treonina	Thr	T
Triptófano	Trp	W
Valina	Val	V

Cuadro 2.1: Lista de aminoácidos estándar.

Un ejemplo de una secuencia de proteína, correspondiente a la proteína 5PNQ [7], es el siguiente:

```
MHHHHHHSSGVDLGTENLYFQSMETMKSKANCAQNPNCNIMIFHPTKEEFNDFDKYIAYMESQG
AHRAGLAKIIPPKEWKARETYDNISEILIATPLQQVASGRAGVFTQYHKKKKAMTVGEYRHLAN
SKKYQTPPHQNFEDLERKYWKNRIYNSPIYGADISGSLFDENTKQWNLGHLGTIQDLLEKECGV
VIEGVNTPYLYFGMWKTTFAWHTEDMDLYSINYLHLGEPKTWYVVPPEHGQRLERLARELFPGS
SRGCGAFLRHKVALISPTVLKENGIPFNRITQEAGEFMVTFPYGYHAGFNHGFNCAEAINFATP
RWIDYGKMASQCSCGEARVTFSMDAFVRILQPERYDLWKRGQDR
```

Adicionalmente y para facilitar el entendimiento estableceremos el siguiente vocabulario para referirnos a cada parte de la proteína que se obtiene durante el algo-

ritmo:

- Secuencia: Para referirnos a una proteína completa.
- Segmento: Parte de la proteína que interactúa con un ligando y recortamos.
- Subsecuencia: Fragmento de la proteína generado por el algoritmo para buscar los patrones.

2.2 Patrones Proteína-Ligando

Un patrón proteína-ligando es una secuencia específica de aminoácidos alrededor de un ligando que se repite en múltiples proteínas. Estos patrones representan regiones funcionales donde ocurre la interacción entre la proteína y la molécula del ligando, lo cual es esencial para la actividad biológica de la proteína. La identificación y análisis de estos patrones permiten comprender mejor cómo las proteínas llevan a cabo sus funciones.

La detección de estos patrones es fundamental en diversas áreas, como el diseño de fármacos, donde se busca identificar moléculas que puedan unirse eficazmente a una proteína objetivo para modificar su actividad. También es crucial en la ingeniería de proteínas, donde modificar o diseñar nuevas proteínas con funciones específicas requiere un entendimiento detallado de los patrones de interacción proteína-ligando.

Para representar estos patrones en secuencias proteicas, se utilizan herramientas y notaciones especializadas como la PA Line de PROSITE[8], que permite describir patrones complejos de manera concisa y estandarizada. Esta notación facilita la comparación y el análisis de secuencias, permitiendo a los investigadores identificar rápidamente regiones de interés y predecir posibles interacciones funcionales.

2.3 Notación PA Line de Prosite

Para representar patrones en secuencias proteicas, se utiliza la notación PA Line de PROSITE[8], una herramienta ampliamente reconocida en bioinformática.

La notación PA Line es un formato estandarizado que permite describir de manera concisa y precisa los patrones de secuencias. Esta notación utiliza símbolos y reglas específicas para representar aminoácidos individuales, opciones alternativas en una

posición dada, repeticiones y rangos de longitud. A continuación, se describen algunos elementos clave de esta notación:[8]:

- **Código IUPAC**: Se utilizan los códigos de una letra estándar para los aminoácidos.
- **Posición indefinida**: Se usa el símbolo `x` para representar cualquier aminoácido en una posición dada.
- **Ambigüedades**: Se indican mediante corchetes `[]` enumerando los aminoácidos permitidos en una posición. Ejemplo: `[ALT]` significa Ala, Leu o Thr. (por las letras A, L y T)
- **Exclusión de aminoácidos**: Se indica mediante llaves {} enumerando los aminoácidos no permitidos en una posición. Ejemplo: {`AM`} significa cualquier aminoácido excepto Ala o Met.
- **Repetición**: Se denota usando un número o un rango entre paréntesis. Ejemplo: `x(3)` representa tres posiciones consecutivas ocupadas por cualquier aminoácido, y `x(2,4)` indica de dos a cuatro posiciones consecutivas de cualquier aminoácido.
- **Restricciones en los extremos**: Se utiliza el símbolo `<` para restringir un patrón al extremo N-terminal y `>` para el extremo C-terminal de una secuencia.
- **Patrón finalizado**: Un patrón siempre termina con un punto (`.`).
- **Ejemplo de patrón**: `PA [AC]-x-V-x(4)-{ED}.` se traduce como: Ala o Cys, seguido de cualquier aminoácido, seguido de Val, seguido de cualquier cuatro aminoácidos, seguido de cualquier aminoácido excepto Glu o Asp.

2.4 Métodos para descubrir patrones

La búsqueda de patrones en secuencias proteicas es una tarea crucial en bioinformática, ya que permite identificar regiones conservadas que pueden ser funcionalmente relevantes, como sitios de unión a ligandos. A continuación, se describen algunos métodos y herramientas clave, incluyendo Pratt y su versión web, y se discute su utilidad para la identificación de patrones proteína-ligando.

2.4.1 Pratt

Pratt es una herramienta ampliamente utilizada para la identificación de patrones conservados en conjuntos de secuencias proteicas. Mediante la búsqueda de patrones comunes en un conjunto de secuencias.

Pratt utiliza algoritmos que consideran la variabilidad en las secuencias, permitiendo identificar patrones que no necesariamente están perfectamente conservados, pero que presentan similitudes significativas. Esto es especialmente útil en el análisis de familias de proteínas donde la conservación es parcial debido a divergencias evolutivas.

Algunas características clave de Pratt incluyen:

- **Flexibilidad en la definición de patrones**: Permite especificar parámetros como la longitud mínima y máxima de los patrones, el número mínimo de secuencias que deben contener el patrón, y la posibilidad de incluir posiciones variables o ambiguas.
- **Incorporación de gaps**: Puede manejar espacios en los patrones, lo que es útil cuando los motivos funcionales están separados por regiones variables.
- **Salida en formato PROSITE**: Los patrones se presentan en la notación PA Line de PROSITE, facilitando su interpretación y uso en otras herramientas bioinformáticas.

2.4.2 Versión Web de Pratt

La versión web de PRATT está disponible a través del servidor ExPASy, que es parte del Swiss Institute of Bioinformatics. Esta interfaz web permite a los usuarios ejecutar Pratt sin necesidad de instalar software localmente, facilitando su acceso y uso.

Además, el Protein Data Bank (PDB) ofrece recursos para el análisis de estructuras proteicas, y aunque no proporciona directamente una versión web de Pratt, las secuencias y estructuras disponibles en el PDB pueden ser utilizadas como entrada para Pratt. Los usuarios pueden extraer secuencias de proteínas del PDB y utilizarlas en Pratt para identificar patrones conservados.

Si bien Pratt es una herramienta potente para identificar patrones conservados en secuencias proteicas, no está específicamente diseñada para identificar patrones

de interacción proteína-ligando. Los patrones descubiertos por Pratt corresponden a secuencias de aminoácidos que están conservadas entre diferentes proteínas, lo que puede incluir sitios de unión a ligandos si estos sitios están conservados.

La identificación de patrones proteína-ligando requiere información sobre las interacciones específicas entre los aminoácidos de la proteína y el ligando. Esto generalmente implica el análisis de datos estructurales 3D, como los disponibles en el PDB, y el uso de herramientas especializadas que consideran la geometría y las propiedades químicas de la interacción.

2.4.3 Método base para descubrir patrones

Este trabajo se inspira en la propuesta secuencial presentada por [9], pero sustituye la generación exhaustiva de subsecuencias por una formulación basada en la *Longest Common Subsequence*. El objetivo sigue siendo identificar patrones compatibles con la notación Prosite, ahora aprovechando la programación dinámica para reducir la complejidad y permitir una paralelización directa.

A continuación, se detalla el flujo del algoritmo para dos cadenas de aminoácidos de largos n y m:

Paso 1: Construcción de la matriz LCS. Se crea una matriz L de tamaño $(n+1) \times (m+1)$ inicializada en cero. Cada celda $L_{i,j}$ representa la longitud de la LCS entre los prefijos $s_{1..i}$ y $t_{1..j}$. El cálculo se realiza mediante la recurrencia descrita anteriormente y tiene complejidad $O(nm)$.

Paso 2: Registro de direcciones. En paralelo con la matriz L se mantiene una matriz D que guarda, para cada celda, las direcciones que preservan la longitud óptima (diagonal, arriba o izquierda). Esta estructura es la base para explorar todas las coincidencias posibles durante el retroceso.

Paso 3: Retroceso y recopilación de alineamientos. Comenzando en la celda (n, m) se sigue la información de D para reconstruir los alineamientos que alcanzan la longitud óptima. Cuando se elige un movimiento diagonal se agrega una coincidencia; los movimientos verticales u horizontales incrementan el conteo de huecos. La exploración puede generar múltiples alineamientos que posteriormente se transforman en patrones distintos.

Paso 4: Formateo y puntuación. Cada alineamiento se traduce a la notación PA Line, colapsando los huecos consecutivos en expresiones `x(n)` y manteniendo en

mayúsculas los aminoácidos conservados. Finalmente, se calcula la puntuación del patrón para ordenar las alternativas.

Ejemplo

Consideremos las cadenas `ABCD` y `AHCD`. La matriz LCS obtenida es:

	$-$	A	H	C	D
$-$	0	0	0	0	0
A	0	1	1	1	1
B	0	1	1	1	1
C	0	1	1	2	2
D	0	1	1	2	3

El valor $L_{4,4} = 3$ indica que la LCS tiene longitud tres. Al retroceder se obtiene el alineamiento `A-H-C-D`, donde `A`, `C` y `D` corresponden a coincidencias y `H` representa un hueco de longitud uno respecto de la primera cadena. A partir de esta información se derivan los siguientes patrones:

- `A-x(1)-C-D`
- `A`
- `C-D`

La misma matriz permite recuperar alineamientos alternativos si existen múltiples trayectorias óptimas, lo que se traduce en patrones adicionales.

Aplicando la función de puntuación con $n = 4$ (longitud máxima de los segmentos analizados) se obtiene:

Patrón	**Puntuación**
A-x(1)-C-D	$1 + 0 \times 4 + 3 \times 4^2 = 49$
A	$0 + 0 \times 4 + 1 \times 4^2 = 16$
C-D	$0 + 0 \times 4 + 2 \times 4^2 = 32$

Cuadro 2.2: Puntuaciones de patrones derivados de la LCS.

El enfoque basado en LCS evita la creación explícita de todas las subcadenas y reduce drásticamente la cantidad de comparaciones necesarias, lo que mejora el

rendimiento y la escalabilidad frente al método exhaustivo. Además, facilita la paralelización del cálculo y del retroceso. Sobre esta base se incorporó el análisis de los residuos que interactúan con cada ligando, de modo que el algoritmo solo procese los segmentos biológicamente relevantes.

3. Método secuencial

3.1 Extracción de datos

Se desarrolló un algoritmo que permite detectar los aminoácidos que interactúan con los ligandos en una proteína y que retorna los segmentos de la proteína que contienen esos aminoácidos.

Para esto se utilizó la librería `Biopython` [1], la cual es una colección de Python disponible gratuitamente módulos para biología molecular computacional, y el paquete Bio.PDB [2] que facilita el uso y procesamiento de los archivos `.pdb`, que corresponden a archivos del Protein Data Bank (PDB) [3].

3.1.1 Descripción del Algoritmo

El algoritmo se compone de varios pasos que se detallan a continuación, junto con ejemplos aplicados a la proteína `1znf` para facilitar su comprensión.

Configuración Inicial

En esta sección, se establecen los parámetros iniciales y se preparan las estructuras de datos necesarias para el análisis.

Algorithm 1 Proceso de análisis de interacción proteína-ligando

1: Establecer el umbral de distancia para considerar una interacción (`distance_threshold`)
2: Leer el archivo PDB para obtener la estructura de la proteína
3: Inicializar listas para residuos de ligando (`ligand_residues`), átomos de proteína (`protein_atoms`) y lista de residuos (`residues_list`)

Ejemplo: Para la proteína `1znf`, se establece un umbral de distancia de 4.0 Å. Se lee el archivo `1znf.pdb` y se inicializan las listas vacías para almacenar los residuos de ligando, átomos de proteína y lista de residuos.

Recopilación de Información de la Estructura

Se recorre la secuencia para identificar los ligandos y aminoácidos de la proteína, almacenando la información relevante en listas para su posterior procesamiento.

Algorithm 2 Proceso de extracción de ligandos y átomos de proteína

```
for all cadena en la estructura do
    for all residuo en la cadena do
        Obtener hetfield, resseq y icode para identificar el residuo // hetfield: Indica el tipo de residuo.
        // resseq: Es el número de secuencia del residuo
        // icode: Código de inserción.
        Obtener el nombre del residuo (resname)
        if el residuo es un ligando then
            Añadir el residuo a ligand_residues
        else if el residuo es de proteína then
            Añadir los átomos del residuo a protein_atoms
            Añadir el residuo a residues_list
        end if
    end for
end for
```

Ejemplo: Al procesar la proteína `1znf`, se identifican los ligandos y de aminoácidos. Los ligandos encontrados son `NH2`, `ACE` y `ZN`, mientras que lo demás corresponde a los aminoácidos de la cadena.

Detección y Procesamiento de Ligandos

Se verifica si existen ligandos en la estructura y se procesa cada uno de ellos individualmente.

Algorithm 3 Identificación de aminoácidos interactuantes con ligandos

1: **if** no se encontraron residuos de ligando **then**
2: Terminar el algoritmo
3: **end if**
4: Obtener códigos únicos de ligandos (`ligand_codes`)
5: **for all** código de ligando `lig_code` en `ligand_codes` **do**
6: Obtener los átomos del ligando actual
7: Proceder con la identificación de aminoácidos interactuantes
8: **end for**

Ejemplo: En la proteína `1znf`, se encontraron los ligandos `NH2`, `ACE` y `ZN`. El algoritmo procesará cada uno de estos ligandos de forma individual.

Identificación de Aminoácidos Interactuantes

Utilizando un algoritmo de búsqueda de vecinos (NeighborSearch), se identifican los aminoácidos de la proteína que interactúan con el ligando, basándose en el umbral de distancia definido. Esto consiste en comparar la distancia de cada átomo del ligando con los átomos de la proteína y determinar si algún átomo del aminoácido está a una distancia menor al umbral. Un residuo se considera interactuante si al menos uno de sus átomos está dentro del umbral de distancia respecto a cualquier átomo del ligando.

Algorithm 4 Búsqueda de residuos interactuantes con ligandos

1: Crear una búsqueda de vecinos (`NeighborSearch`) con los átomos de proteína
2: Inicializar un conjunto de residuos que interactúan (`interacting_residues`)
3: **for all** átomo en los átomos del ligando **do**
4: Buscar residuos vecinos dentro del umbral de distancia
5: Añadir estos residuos a `interacting_residues`
6: **end for**

Ejemplo: Identificación de Aminoácidos Interactuantes Al procesar el ligando `NH2` de la proteína `1znf`, el algoritmo identifica los aminoácidos que se encuentran dentro de 4.0 Å de cualquier átomo del ligando. Los aminoácidos identificados corresponden a los aminoácidos `H`, `K` y `N` al final de la secuencia.

Generación de la Secuencia Modificada

Se genera la secuencia completa de aminoácidos de la proteína, utilizando letras mayúsculas para los aminoácidos que interactúan con el ligando y minúsculas para los que no.

Algorithm 5 Generación de secuencia resaltada de residuos

```
Inicializar la secuencia resaltada (highlighted_sequence)
for all residuo en residues_list do
    Obtener el nombre del residuo (resname)
    Convertir resname a código de una letra (one_letter)
    if el residuo está en interacting_residues then
        Convertir one_letter a mayúscula
    else
        Convertir one_letter a minúscula
    end if
    Añadir one_letter a highlighted_sequence
end for
```

Ejemplo: Generación de la Secuencia Modificada Con la secuencia original de la proteína `1znf`:

`ykcglcersfveksalsrhqrvhkn`

Y considerando que los residuos que interactúan con el ligando `NH2` son `H`, `K` y `N`, la secuencia modificada será:

`ykcglcersfveksalsrhqrvHKN`

Las letras mayúsculas indican los aminoácidos que interactúan con el ligando.

Extracción y Presentación del Segmento Interactuante

Se extrae el segmento de la secuencia que abarca desde el primer hasta el último aminoácido que interactúa con el ligando.

Algorithm 6 Detección de segmento interactuante en la secuencia resaltada

1: Utilizar una expresión regular para encontrar el segmento que inicia y termina con una letra mayúscula
2: **if** se encuentra una coincidencia **then**
3: Extraer el segmento interactuante (`interacting_segment`)
4: Mostrar el segmento interactuante al usuario
5: **else**
6: Indicar que no se encontraron residuos que interactúen con el ligando
7: **end if**

ligando y se muestra al usuario.

Algorithm 7 Detección de segmento interactuante en la secuencia resaltada

1: Utilizar una expresión regular para encontrar el segmento que inicia y termina con una letra mayúscula
2: **if** se encuentra una coincidencia **then**
3: Extraer el segmento interactuante (`interacting_segment`)
4: Mostrar el segmento interactuante al usuario
5: **else**
6: Indicar que no se encontraron residuos que interactúen con el ligando
7: **end if**

Ejemplo: Extracción y Presentación del Segmento Interactuante Aplicando la expresión regular a la secuencia modificada:

`ykcglcersfveksalsrhqrvHKN`

Se extrae el segmento que va desde la primera letra mayúscula hasta la última:

`HKN`

Este es el segmento de aminoácidos que interactúa con el ligando `NH2`. El mismo proceso se repite para los otros ligandos identificados.

3.1.2 Construcción de la matriz LCS

Para cada par de segmentos que interactúan con un ligando se construye una matriz $(n+1)\times(m+1)$ que almacena la longitud de la *Longest Common Subsequence* entre los prefijos de ambas cadenas. La recurrencia empleada es la usual en programación dinámica:

$$L_{i,j} = \begin{cases} L_{i-1,j-1} + 1 & \text{si } s_i = t_j, \\ \max(L_{i-1,j}, L_{i,j-1}) & \text{en otro caso.} \end{cases}$$

Además de la matriz L se mantiene una matriz de direcciones D que registra si la mejor transición proviene de la diagonal, desde arriba o desde la izquierda. Esta información permite reconstruir posteriormente todos los alineamientos que alcanzan la longitud óptima.

Para los segmentos de ejemplo `AbC` y `AgC`, la matriz resultante es:

	$-$	A	g	C
$-$	0	0	0	0
A	0	1	1	1
b	0	1	1	1
C	0	1	1	2

El valor $L_{3,3} = 2$ indica que la LCS tiene longitud dos y que existen coincidencias suficientes para derivar un patrón candidato útil.

3.1.3 Recuperación de patrones candidatos

La reconstrucción de los patrones se realiza mediante un recorrido inverso sobre D. Cada vez que existen múltiples direcciones posibles se exploran todas las alternativas para obtener alineamientos diferentes. El proceso genera secuencias intermedias en las que se alternan coincidencias exactas con huecos que representan los aminoácidos no compartidos entre ambas cadenas.

En el ejemplo anterior, el retroceso toma una diagonal al encontrar la coincidencia `C`, avanza hacia arriba para contabilizar un hueco de longitud uno y finaliza con la coincidencia `A`. El alineamiento resultante se escribe:

- `A-C`

3.1.4 Formateo de patrones

Cada alineamiento se transforma a la notación PA Line de Prosite. Los aminoácidos conservados se expresan con letras mayúsculas y los huecos se convierten en expresiones `x(n)`, donde n es la longitud del segmento ausente. Utilizando el alineamiento previo se obtiene:

- `A-g-C` $\rightarrow$ `A-x(1)-C`
- `A` $\rightarrow$ `A`
- `C` $\rightarrow$ `C`

Además, se define una notación complementaria para los casos en que un patrón presenta segmentos ausentes de longitudes X_i observadas pero no de una longitud X_j determinada. En tales situaciones se registran explícitamente las longitudes detectadas; por ejemplo, la notación

`A-x(1|2|3|5)-C`

indica que el patrón admite un hueco de longitud 1, 2, 3 o 5, y que no se ha observado un hueco de longitud 4.

3.1.5 Asignación de Puntuaciones

A cada patrón se le asigna una puntuación basada en la siguiente fórmula:

$$\text{Puntuación} = x + m \times n + y \times n^2$$

Donde:

- x: Número de `x` (gaps) en el patrón.
- m: Número de letras minúsculas en el patrón.
- y: Número de letras mayúsculas en el patrón.
- n: Longitud del segmento original.

Para nuestro ejemplo, con segmentos de longitud $n = 3$, calculamos las puntuaciones:

- `A-x(1)-C`:
  - $x = 1$ (un gap)
  - $m = 0$ (cero minúsculas)
  - $y = 2$ (A y C son mayúsculas)
  - Puntuación: $1 + 0 \times 3 + 2 \times 3^2 = 1 + 0 + 18 = 19$
- `A`:
  - $x = 0$
  - $m = 0$
  - $y = 1$
  - Puntuación: $0 + 0 \times 3 + 1 \times 3^2 = 0 + 0 + 9 = 9$
- `C`:
  - $x = 0$
  - $m = 0$
  - $y = 1$
  - Puntuación: $0 + 0 \times 3 + 1 \times 3^2 = 0 + 0 + 9 = 9$

3.1.6 Ordenamiento de Patrones

Finalmente, se ordenan los patrones según su puntuación de mayor a menor:

1. `A-x(1)-C` (19 puntos)
2. `A` (9 puntos)
3. `C` (9 puntos)

Siendo el patrón A-x(1)-C el elegido como mejor patrón.

3.2 Implementación

El algoritmo basado en LCS para descubrir patrones proteína-ligando se implementó en tres etapas: construcción de la matriz de programación dinámica, recuperación y formateo de alineamientos. Cada etapa expone puntos de paralelización que permiten distribuir el trabajo entre hilos independientes, ya sea dividiendo la matriz por diagonales o repartiendo los caminos de retroceso entre colas de tareas.

3.2.1 Construcción de la matriz LCS

La primera fase crea las matrices de longitud y de direcciones necesarias para el retroceso posterior. El cálculo se realiza fila a fila respetando las dependencias de la programación dinámica; en la versión paralela se procesan las diagonales de la matriz en paralelo para maximizar el uso de los núcleos disponibles.

Algorithm 8 ConstruirMatricesLCS

```
1: function ConstruirMatricesLCS(secuenciaA, secuenciaB)
2:     n ← len(secuenciaA)
3:     m ← len(secuenciaB)
4:     L ← matriz(n + 1, m + 1, 0)
5:     D ← matriz(n + 1, m + 1, ∅)
6:     for i ← 1 to n do
7:         for j ← 1 to m do
8:             if secuenciaA[i − 1] = secuenciaB[j − 1] then
9:                 L[i][j] ← L[i − 1][j − 1] + 1
10:                D[i][j].add(DIAGONAL)
11:            else
12:                arriba ← L[i − 1][j]
13:                izquierda ← L[i][j − 1]
14:                mejor ← max(arriba, izquierda)
15:                L[i][j] ← mejor
16:                if arriba = mejor then
17:                    D[i][j].add(ARRIBA)
18:                end if
19:                if izquierda = mejor then
20:                    D[i][j].add(IZQUIERDA)
21:                end if
22:            end if
23:        end for
24:    end for
25:    return (L, D)
26: end function
```

3.2.2 Recuperación y formateo de patrones

Una vez calculadas las matrices, se realiza el retroceso para reconstruir los alineamientos que alcanzan la longitud óptima. El procedimiento explora todas las trayectorias registradas en D y traduce cada alineamiento en una cadena compatible con la notación PA Line, colapsando los huecos consecutivos en expresiones `x(n)`.

En la implementación paralela, cada hilo procesa una rama diferente del árbol de retroceso para generar los patrones de forma concurrente.

Algorithm 9 RecuperarYFormatearPatrones

```
1: function RecuperarYFormatearPatrones(D, secuenciaA, secuenciaB)
2:     n ← len(secuenciaA)
3:     m ← len(secuenciaB)
4:     patrones ← conjunto vacío
5:     Backtrack(n, m, [])
6:     return patrones
7: end function
8: procedure Backtrack(i, j, alineacion)
9:     if i = 0 or j = 0 then
10:        patron ← Formatear(alineacion)
11:        if patron ≠ ∅ then
12:            patrones.add(patron)
13:        end if
14:        return
15:    end if
16:    for mov ∈ D[i][j] do
17:        if mov = DIAGONAL then
18:            Backtrack(i − 1, j − 1, [(secuenciaA[i − 1], secuenciaB[j − 1])] + alineacion)
19:        else if mov = ARRIBA then
20:            Backtrack(i − 1, j, [(secuenciaA[i − 1], −)] + alineacion)
21:        else
22:            Backtrack(i, j − 1, [(−, secuenciaB[j − 1])] + alineacion)
23:        end if
24:    end for
25: end procedure
26: function Formatear(alineacion)
27:    resultado ← []
28:    gap ← 0
29:    tieneResiduo ← falso
30:    for (a, b) ∈ alineacion do
31:        if a = b and a es mayúscula then
32:            if gap > 0 then
33:                resultado.add(x(gap))          ▷ Se colapsan huecos consecutivos
34:                gap ← 0
35:            end if
36:            resultado.add(a)
```

3.2.3 Asignación de Puntuaciones

Cada patrón recuperado se evalúa mediante la misma función de puntuación empleada en la sección anterior. La fórmula toma como base la cantidad de huecos, letras minúsculas y letras mayúsculas:

$$\text{Puntuación} = x + m \times n + y \times n^2$$

Algorithm 10 Asignar puntuaciones y ordenar

```
1: function OrdenarYAsignarPuntuaciones(patrones, longitudSegmento)
2:     puntuaciones ← []
3:     for patrón ∈ patrones do
4:         puntuación ← calcularPuntuación(patrón, longitudSegmento)
5:         puntuaciones.add(puntuación)
6:     end for
7:     return ordenarPorPuntuación(patrones, puntuaciones)
8: end function
```

3.3 Experimentos

El algoritmo basado en LCS demostró ser efectivo para identificar y resaltar patrones comunes entre dos secuencias de aminoácidos, permitiendo detectar similitudes y variaciones significativas sin incurrir en la explosión combinatoria del enfoque exhaustivo. Al operar sobre la matriz de programación dinámica y reutilizar los alineamientos recuperados, es posible analizar con mayor profundidad las estructuras de las secuencias y ordenar los patrones según su relevancia, lo cual resulta útil en bioinformática, análisis de datos y procesamiento de lenguaje natural.

4. Método en paralelo

4.1 Método 1

4.1.1 Diseño

4.1.2 Implementación

4.2 Método 2

4.2.1 Diseño

4.2.2 Implementación

4.3

5. Conclusiones

6. Anexos

Glosario

El primer término: Este es el significado del primer término, realmente no se bien lo que significa pero podría haberlo averiguado si hubiese tenido un poco mas de tiempo.

El segundo término: Este si se lo que significa pero me da lata escribirlo...

Bibliografía

[1] [**Pendiente**] Referencia real para Biopython. Reemplazar este placeholder por la cita correcta.

[2] [**Pendiente**] Referencia real para Bio.PDB. Sustituir por la fuente correspondiente.

[3] [**Pendiente**] Fuente oficial del Protein Data Bank (PDB). Actualizar con la referencia definitiva.

[4] [**Pendiente**] Definición formal de aminoácidos. Cambiar por la cita verificada.

[5] [**Pendiente**] Definición formal de proteínas. Cambiar por la cita verificada.

[6] [**Pendiente**] Definición detallada de ligandos. Reemplazar por referencia final.

[7] [**Pendiente**] Información sobre la proteína 5PNQ. Sustituir por la fuente correcta.

[8] [**Pendiente**] Manual o documentación de la notación PA Line de PROSITE. Reemplazar con la referencia adecuada.

[9] [**Pendiente**] Trabajo previo de Francisco que sirve de base. Actualizar con la cita oficial.

ANEXOS

A. El Primer Anexo

Aquí va el texto del primer anexo...

A.1 La primera sección del primer anexo

Aquí va el texto de la primera sección del primer anexo...

A.2 La segunda sección del primer anexo

Aquí va el texto de la segunda sección del primer anexo...

A.2.1 La primera subsección de la segunda sección del primer anexo

B. El segundo Anexo

Aquí va el texto del segundo anexo...

B.1 La primera sección del segundo anexo

Aquí va el texto de la primera sección del segundo anexo...