UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

Predicción de la Estructura de Proteínas mediante Optimización Multiobjetivo

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería de Computadores

Trabajo Fin de Grado

Predicción de la Estructura de Proteínas mediante Optimización Multiobjetivo

Autor: Alberto Rici Vázquez

Tutor: Miguel Ángel Vega Rodríguez

ÍNDICE GENERAL DE CONTENIDOS

Resumen

En el siguiente trabajo de fin de grado presentamos un algoritmo multiobjetivo mediante el cual podremos predecir la estructura terciaria de las proteínas. Este algoritmo está basado en computación evolutiva y bio-inspirada, y se apoya en el cálculo de un conjunto de campos de fuerzas ampliamente utilizado en mecánica y dinámica molecular.

En nuestro caso hemos usado una función de energía llamada CHARMM (*Chemistry at HARvard Macromolecular Mechanics*). Para poder utilizar esta función de energía se ha utilizado un software externo a nuestro trabajo, Tinker. Este software está compuesto por una gran cantidad de módulos de mecánica y dinámica molecular. Además, está escrito en lenguaje Fortran 77 con algunas extensiones de C.

Además de haber realizado este algoritmo multiobjetivo, lo hemos paralelizado. Dado que hemos realizado los experimentos en un sistema multiprocesador y multicore, hemos utilizado OpenMP como entorno de paralelización. Pero no ha acabado aquí todo nuestro trabajo de paralelización, sino que debido a que Tinker utilizaba los componentes de entrada y salida de nuestro equipo de trabajo, tuvimos que solucionar estos conflictos para que se pudiera realizar la paralelización con éxito.

En definitiva, hemos implementado un algoritmo evolutivo multiobjetivo paralelizado, en nuestro caso basado en el comportamiento de las abejas (inteligencia de enjambre). Y junto a los módulos que necesitamos de Tinker, pretendemos conseguir datos adecuados de predicción de la estructura de las proteínas.

Los resultados que hemos obtenido a priori son positivos, dado que energéticamente obtenemos mejores resultados que otros autores y respecto a la disposición espacial de las proteínas, obtenemos resultados competitivos.

1. Introducción

En este apartado se comentarán los objetivos que se plantearon al principio de la selección de este trabajo final de grado y después se irán recorriendo los demás apartados de los que consta esta documentación, explicando brevemente cada uno de ellos.

1.1. Objetivos

Los objetivos que se plantearon alcanzar con este trabajo son los siguientes:

- Predicción de la estructura terciaria de las proteínas.
- Esta predicción realizarla mediante técnicas de optimización multiobjetvo.
- Todo esto mediante un algoritmo evolutivo. En nuestro caso se eligió el MOABC (*Multiobjective artificial bee colony*).
- Dado que la predicción de las proteínas es un proceso computacionalmente costoso, estudiar su coste temporal e intentar reducirlo, a través del paralelismo.
- Y esto siempre obteniendo resultados competitivos a nivel de la optimización multiobjetivo, es decir, obtener buenos resultados energéticos.

1.2. Estructura de la documentación

A continuación, hablaremos de cada uno de los grandes apartados que componen el resto de la memoria.

En el apartado 2, proteínas, conceptos fundamentales, introduciremos los conceptos más importantes sobre las proteínas, centrándonos principalmente en los diferentes tipos de estructuras y sus características.

El siguiente apartado, el 3, se centra en la predicción de la estructura terciaria de las proteínas. Comentaremos en qué consiste esta tarea y cuáles son sus precedentes y los avances que se han hecho en este campo.

El apartado 4 lleva por título “técnicas de optimización”. Aquí introduciremos de forma teórica los diferentes tipos y diferencias entre las distintas técnicas de optimización. Principalmente nos centraremos en las técnicas de inteligencia de enjambre.

Como ya comentamos en los objetivos, realizaremos una predicción basada en optimización multiobjetivo. Por lo que el apartado 5, optimización multiobjetivo, nos introducirá los conceptos fundamentales sobre ésta.

También hemos comentado la necesidad de aplicar paralelismo, en el apartado 6 se incluye una introducción al mismo, explicando los conceptos fundamentales y la necesidad del paralelismo. Además, también hablaremos de la API que se ha utilizado en este trabajo (OpenMP).

En el siguiente apartado 7, hardware y software utilizados, comentaremos qué tipo de recursos hardware hemos utilizado para la ejecución de nuestro algoritmo y qué software externo se ha utilizado.

En este apartado 8, MOABC, comentaremos las características y la forma que tiene el diseño de nuestro algoritmo.

En el siguiente apartado, el 9, nos centraremos en la paralelización de nuestra propuesta. Hablaremos sobre la forma en la que se realizó el estudio del impacto del tiempo de ejecución y qué zonas son las que más tiempo de cómputo consumen. Y tras este estudio, se detallará la forma en la que aplicamos paralelismo y cómo resolvimos los problemas con los que nos fuimos encontrando.

En el apartado 10, ajuste del algoritmo, comentaremos todas las pruebas que hemos realizado para ajustar de la forma más óptima el algoritmo.

Los dos siguientes apartados, 11 y 12, son los apartados donde se hablará de los resultados finales obtenidos. En el primero, el apartado 11, se detallan los resultados de la paralelización, realizando un estudio de la eficiencia y *speedup* (aceleración). En el otro apartado, el 12, se habla de los resultados biológicos y multiobjetivo (energéticos) que hemos obtenido al aplicar nuestra propuesta a diferentes proteínas.

En el siguiente apartado, el 13, comentaremos las diferencias que hay entre nuestros resultados y los de otros algoritmos propuestos por otros autores.

El penúltimo apartado es el 14, conclusiones. Estableceremos las conclusiones de si hemos alcanzado nuestros objetivos y de qué forma. Además, hablaremos de qué posibles trabajos futuros podríamos aplicar a este trabajo final de grado.

Por último, en el apartado 15, referencias bibliográficas, pondremos de donde se han sacado las ideas sobre los conceptos que exponemos en este trabajo.

2. Proteínas, conceptos fundamentales

En este apartado comentaremos los conceptos más importantes sobre las proteínas, esta información se ha obtenido de (del Pozo, 2011). Las proteínas son las macromoléculas más abundantes en las células, más del 50% del peso seco de la célula son proteínas. Están constituidas, fundamentalmente, por C (carbono), H (hidrógeno), O (oxígeno) y N (nitrógeno). Este último es el bioelemento más característico de las proteínas. En la ilustración 1 se puede observar un ejemplo de estructura de proteína.

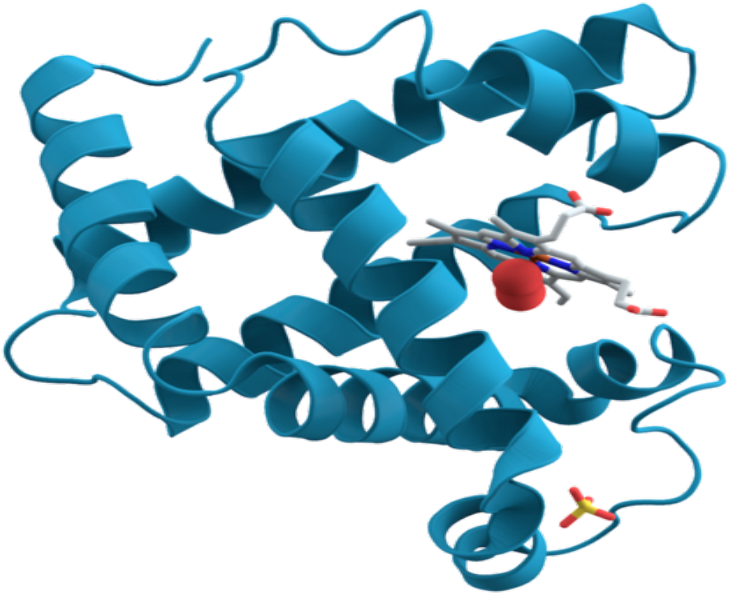

Ilustración 1. Estructura proteica tridimensional de la mioglobina. Esta imagen está sacada de https://es.wikipedia.org/wiki/Prote%C3%ADna

Las proteínas están formadas por aminoácidos (ilustración 2). Hay sólo 20 aminoácidos diferentes (tabla 1), y la unión entre ellos se realiza mediante enlaces peptídicos.

Además, a través de las proteínas se expresa la información genética (ADN y ARN). Todos los aminoácidos, salvo la prolina, responden a una fórmula general.

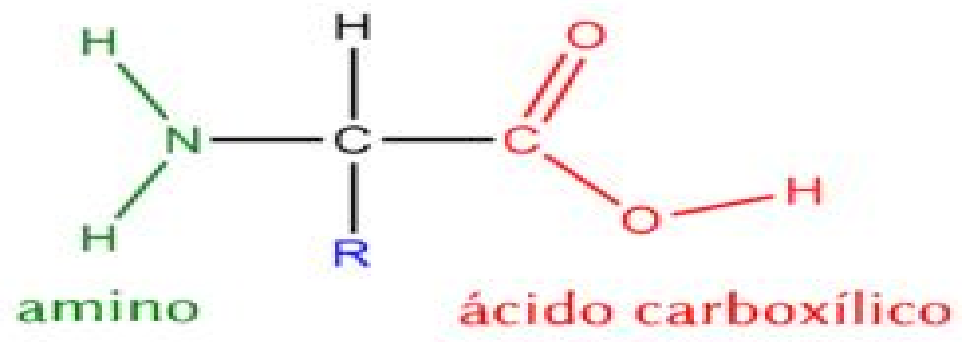

Ilustración 2. Fórmula general del aminoácido. Esta imagen está sacada de https://es.wikipedia.org/wiki/Amino%C3%A1cido

La mayoría de los aminoácidos pueden sintetizarse unos a partir de otros, pero existen otros, aminoácidos esenciales, que no pueden ser sintetizados por el organismo y deben obtenerse en la dieta habitual. Los aminoácidos esenciales son diferentes para cada especie, en la especie humana los aminoácidos esenciales son 10, por ejemplo, triptófano o metionina, entre otros.

*Tabla 1. Listado de los 20 aminoácidos de los seres vivos, los marcados con * son los aminoácidos esenciales.*

Glicina	Treonina*	Alanina	Tirosina	Valina*
Cisteína	Isoleuciona*	Glutamina	Metionina*	Asparagina
Prolina	Áspartato	Leucina*	Glutamato	Fenilalanina*
Lisina*	Triptófano*	Arginina*	Serina	Histidina*

Cuando reacciona el grupo ácido de un aminoácido con el grupo amino de otro, ambos aminoácidos quedan unidos mediante un enlace peptídico. Éste es un enlace covalente que se establece entre los átomos de carbono y nitrógeno. Es un enlace muy resistente, lo que hace posible el gran tamaño y estabilidad de las moléculas proteicas.

Las proteínas en condiciones normales poseen solamente una conformación y ésta es la responsable de la función que realiza cada una de ellas. La compleja estructura de las proteínas puede estudiarse a diferentes niveles (ilustración 8).

2.1. Estructura primaria

La poseen todas las proteínas y viene dada por el orden o la secuencia de los aminoácidos de las proteínas (ilustración 3). Esta secuencia es la que determina el resto de los niveles y como consecuencia la función de la proteína.

La alteración de la estructura primaria por eliminación, adicción o intercambio de los aminoácidos puede cambiar la configuración general de una proteína y dar lugar a una proteína diferente.

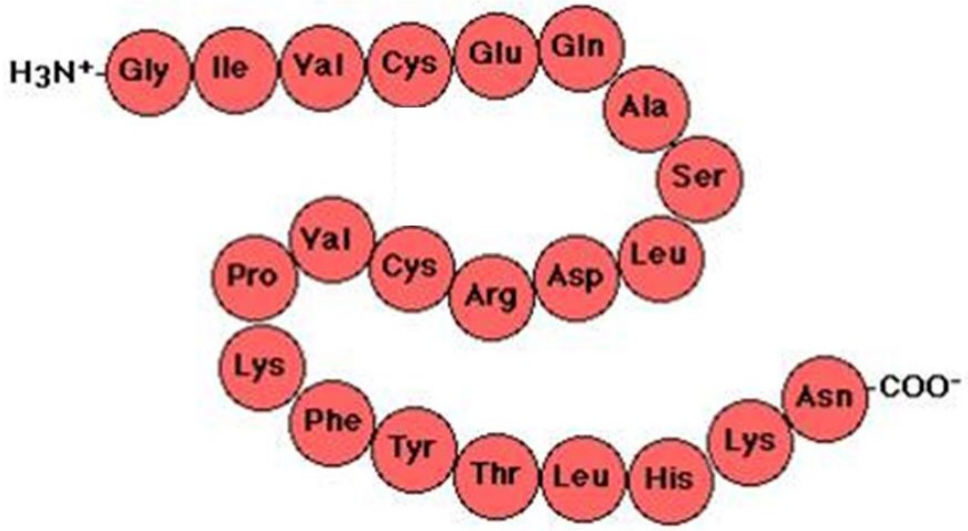

Ilustración 3. Estructura primaria de una proteína. Esta imagen está sacada de http://slideplayer.es/slide/7234472/

2.2. Estructura secundaria

Es la disposición espacial que presenta la secuencia de aminoácidos (estructura primaria) para ser estable. Una de las características de los enlaces peptídicos es que imponen restricciones que obligan a que las proteínas adopten una determinada estructura secundaria. Ésta puede ser principalmente de dos tipos.

2.2.1. Estructura secundaria en hélice alfa

En este tipo de estructuras, la secuencia de aminoácidos se va enrollando sobre sí misma adoptando una disposición helicoidal (ilustración 4). Esta espiral se mantiene gracias a los puentes de hidrógeno intracatenarios.

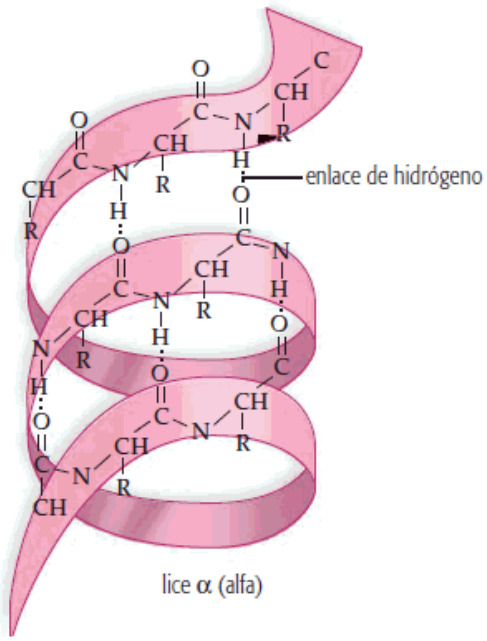

Ilustración 4. Estructura secundaria en hélice alfa de una proteína. Esta imagen está sacada de http://www.guatequimica.com/tutoriales/proteinas/Estructura_de_las_proteinas.htm?iframe=true&width=95%&height=95%

2.2.2. Estructura secundaria en lámina plegada beta

En este caso se producen asociaciones dentro de la cadena como si se tratasen de cadenas diferentes (ilustración 5), dando forma de una lámina. Se establecen entonces uniones mediante puentes de hidrógeno intercatenarios.

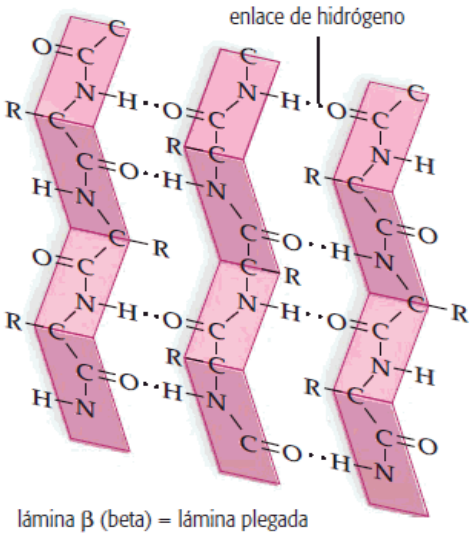

Ilustración 5. Estructura secundaria en lámina plegada beta de una proteína. Esta imagen está sacada de http://www.guatequimica.com/tutoriales/proteinas/Estructura_de_las_proteinas.htm?iframe=true&width=95%&height=95%

2.3. Estructura terciaria

La estructura terciaria se da cuando adoptan una estructura espacial tridimensional (ilustración 6) y su estabilidad es gracias a las siguientes interacciones: puentes de hidrógeno, puentes de disulfuro, atracciones electrostáticas y atracciones hidrofóbicas.

Las características y las funciones de una proteína dependen de la estructura terciaria que tenga, por ello ésta es específica para cada proteína.

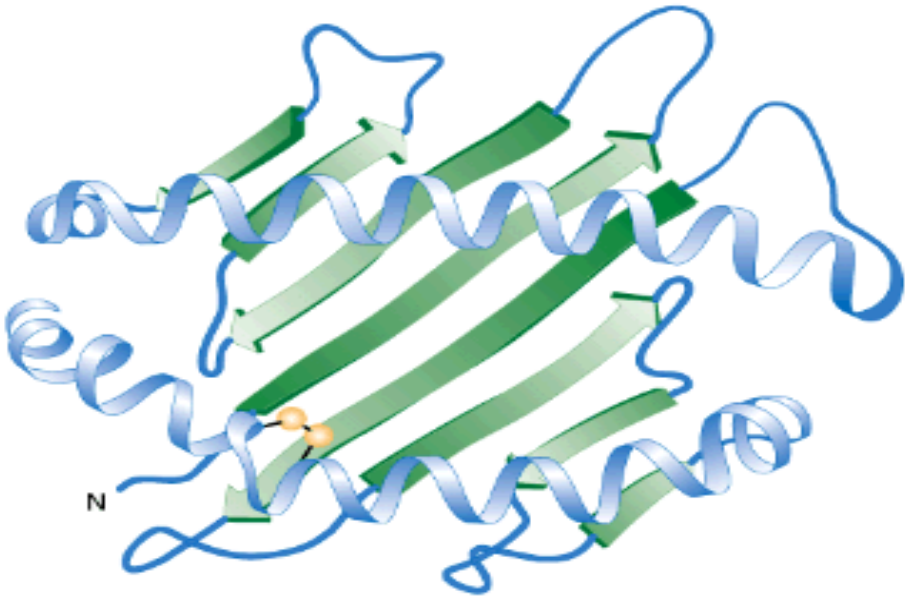

Ilustración 6. Estructura terciaria de una proteína, azul: hélice alfa, verde: lámina plegada. Esta imagen está sacada de https://lacienciaysusdemonios.com/2010/05/04/evolucion-molecular-1-introduccion-i-proteinas-las-piezas-del-reloj/

2.4. Estructura cuaternaria

Cuando varias cadenas de aminoácidos con estructura terciaria se unen para formar una proteína, es lo que llamamos estructura cuaternaria (ilustración 7). Los enlaces entre cada una de las cadenas son enlaces débiles (no covalentes).

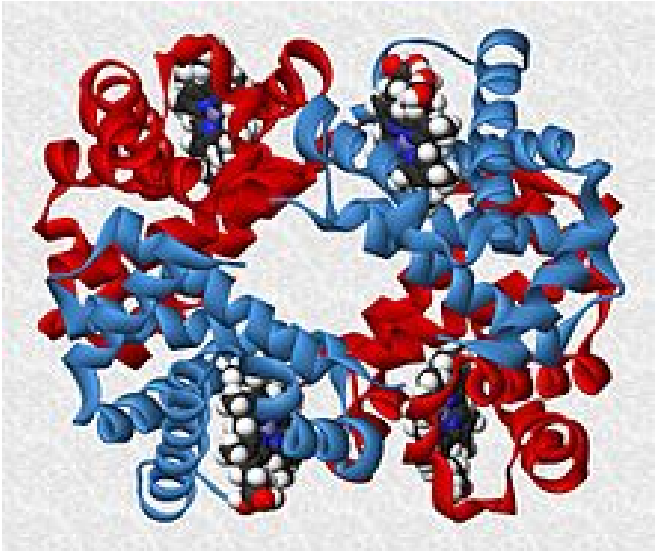

Ilustración 7. Estructura cuaternaria de una proteína. Esta imagen está sacada de https://es.wikipedia.org/wiki/Estructura_cuaternaria_de_las_prote%C3%ADnas

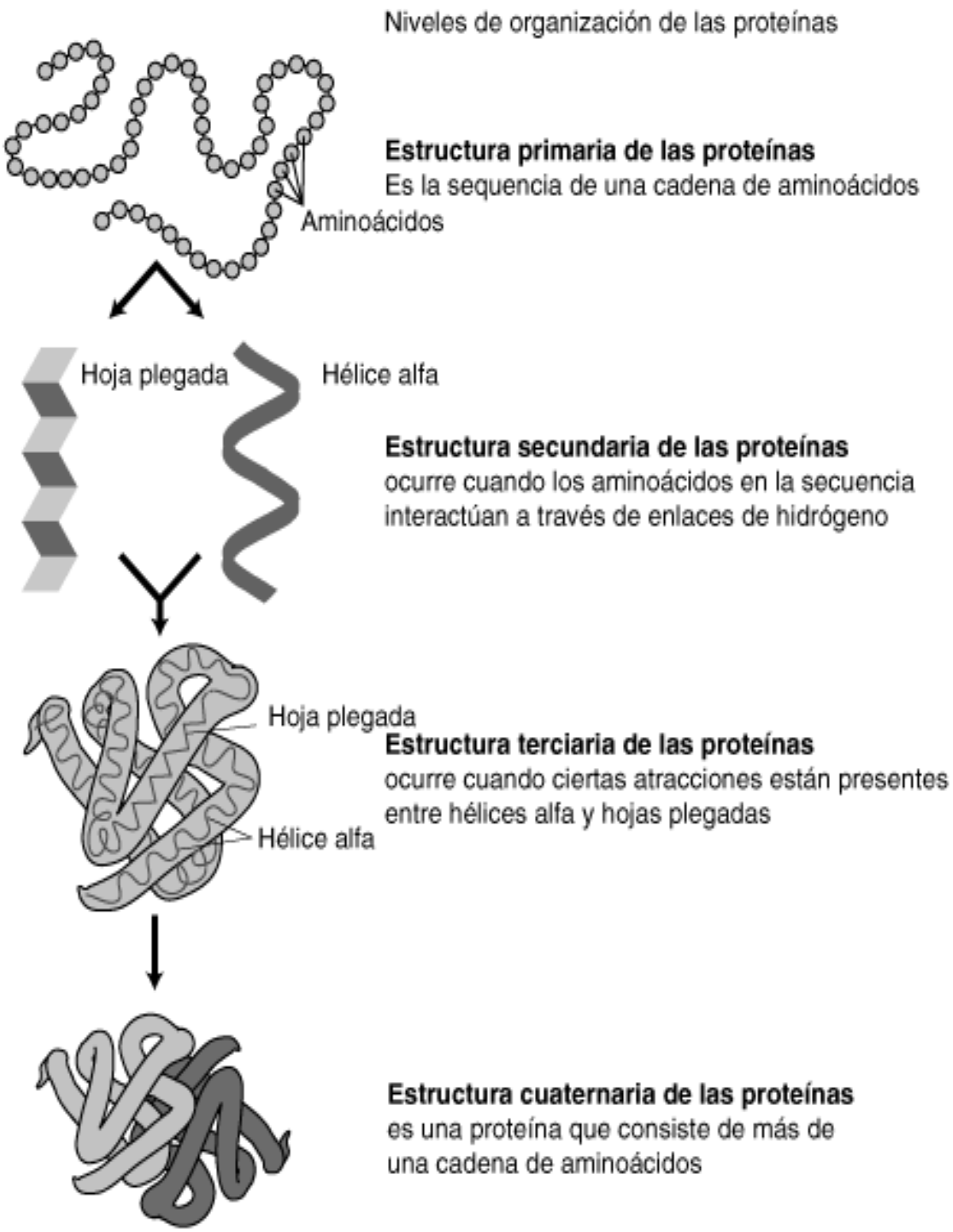

Ilustración 8. Niveles de estructura de las proteínas. Esta imagen está sacada de https://temasdebioquimica.wordpress.com/2008/10/01/estructura-primaria-de-las-proteinas/

3. Predicción de la estructura terciaria de las proteínas

Este apartado se divide en dos secciones. En la primera explicamos los conceptos fundamentales sobre la predicción de la estructura terciaria de las proteínas. En la segunda sección nos centramos en cómo saber (qué métricas usar para saber) la calidad de la predicción realizada.

3.1. Conceptos fundamentales

Aquí comentaremos el concepto de la predicción de la estructura de las proteínas, esta información ha sido obtenida de varias fuentes, pero principalmente de los artículos de (Cutello et al., 2006) y (Calvo et al., 2011b).

El problema de la predicción de la estructura de la proteína (*Protein Structure Prediction*, PSP), en nuestro caso, hace referencia a la predicción de la estructura terciaria nativa, doblada de una proteína dada su secuencia de aminoácidos. Las funciones energéticas potenciales utilizadas en la literatura para evaluar la conformación de una proteína se basan en los cálculos de dos energías de interacción diferentes: locales (átomos de enlace, *bond*) y no locales (átomos de no-enlace, *non-bond*).

El problema de la PSP es actualmente uno de los problemas más desafiantes en bioinformática y bioquímica. Se define simplemente como la tarea de comprender y predecir cómo la información codificada en la secuencia de aminoácidos de las proteínas se traduce en la estructura tridimensional de la proteína biológicamente activa.

Si pudiéramos resolver el problema de la PSP simplificaríamos enormemente, por ejemplo, la tarea de comprender el mecanismo de las enfermedades hereditarias e infecciosas, de diseñar fármacos con propiedades terapéuticas específicas y de crear polímeros biológicos con propiedades materiales específicas.

El plegamiento de proteínas (*Protein Folder*, PF) es distinto de PSP. En el PSP, no nos interesa el proceso de plegado, sino sólo la estructura tridimensional final obtenida. Los métodos comunes para la búsqueda de la estructura de las proteínas (como la radiografía de rayos X y RMN-resonancia magnética nuclear) son lentos y costosos, y pueden tomar hasta varios meses de trabajo de laboratorio. Como consecuencia, ha habido un creciente interés en el diseño de algoritmos para el problema PSP.

La predicción exitosa de la estructura de una proteína requiere una función de energía libre suficientemente cercana al potencial correcto para el estado nativo, así como un método para explorar el espacio conformacional. Las funciones de energía potencial actuales, sin embargo, tienen exactitud limitada y el espacio conformacional es inmenso.

Varios enfoques algorítmicos han sido aplicados al problema de PSP en los últimos 20 años: dinámica molecular, recocido (o enfriamiento) simulado, algoritmos evolutivos, etc. A pesar de todos estos esfuerzos, el PSP sigue siendo un desafiante problema computacionalmente abierto.

3.2. Evaluación de la calidad de la predicción

A continuación, nos centraremos más en la forma de evaluación de las estructuras de las proteínas, concretamente la que se ha decidido utilizar para este trabajo final de grado. Para evaluar la estructura de una proteína necesitamos utilizar algunas funciones de energía.

Para llegar a algunas buenas funciones de energía sería natural utilizar la mecánica cuántica, pero es demasiado complejo computacionalmente para ser práctico en la modelización de sistemas más grandes, por lo que usamos la física clásica. A veces denominadas funciones de energía potencial o campos de fuerza, estas funciones devuelven un valor para la energía basado en la conformación de la proteína. Proporcionan información sobre qué conformaciones de la proteína son mejores o

peores. Cuanto menor sea el valor energético, mejor será la conformación. Las funciones energéticas más típicas tienen la forma de la ecuación 1.

Ecuación 1. Fórmula general de las funciones de energía

$$E(R) = \sum_{bonds} B(R) + \sum_{angles} A(R) + \sum_{torsions} T(R) + \sum_{non-bonded} N(R)$$

Donde R es el vector que representa la conformación de la proteína, típicamente en coordenadas cartesianas o en ángulos de torsión.

En este trabajo, con el fin de evaluar la conformación de una proteína, se utiliza la función de energía de la química en la mecánica macromolecular de la Universidad de Harvard (*Chemistry at HARvard Macromolecular Mechanics*, CHARMM) (v.27). CHARMM es un campo de fuerza popular, usado principalmente para estudiar (tener en cuenta) todos los átomos. Es una suma compuesta de varias ecuaciones de mecánica molecular. La función de energía CHARMM tiene la forma de la ecuación 2.

Ecuación 2. Fórmula de la energía de CHARMM

$$E_{charmm} = \sum_{bonds} k_b(b - b_0)^2 + \sum_{UB} k_{UB}(S - S_0)^2 + \sum_{angles} k_\theta(\theta - \theta_0)^2 + \sum_{torsions} k_\chi[1 + \cos(n\chi - \delta)] + \sum_{impropers} k_{IMP}(\phi - \phi_0)^2 + \sum_{non-bond} \varepsilon_{ij}\left[\left(\frac{Rmin_{ij}}{r_{ij}}\right)^{12} - \left(\frac{Rmin_{ij}}{r_{ij}}\right)^{6}\right] + \frac{q_i q_j}{e r_{ij}},$$

Donde:

- b es la longitud del enlace, b_0 es la distancia de equilibrio del enlace y k_b es la fuerza de enlace constante.
- S es la distancia entre dos átomos separados por dos enlaces covalentes (distancia 1, 3), S_0 es la distancia de equilibrio y k_{UB} es la constante de fuerza Urey Bradley.

- θ es el ángulo de valencia, θ_0 es el ángulo de equilibrio y k_θ es la fuerza de ángulo de valencia constante.
- χ es el diedro o ángulo de torsión, k_χ es la fuerza diédrica constante, n es la multiplicidad y δ es el ángulo de fase.
- ϕ es el ángulo impropio, ϕ_0 es el ángulo de equilibrio inadecuado y k_{IMP} es la constante de fuerza inadecuada.
- ε_{ij} es la profundidad de Lennard Jones, r_{ij} es la distancia entre los átomos i y j, $Rmin_{ij}$ es el radio mínimo de interacción, q_i y q_j son las cargas atómicas parciales y e es la constante dieléctrica.

Típicamente, ε_i y $Rmin_i$ se obtienen para tipos de átomos individuales y luego se combinan para producir ε_{ij} y $Rmin_{ij}$ para los átomos que interactúan mediante reglas de combinación. En CHARMM, los valores ε_{ij} se obtienen a través de la media geométrica $\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j}$, y $Rmin_{ij}$ a través de la media aritmética $Rmin_{ij} = (Rmin_i + Rmin_j)/2$.

Pero esta función determina la energía de la proteína predicha, ahora toca evaluar la similitud de la conformación de dicha proteína. Para ello se ha decidido emplear la desviación cuadrática media de la raíz (*Root Mean Square Deviation*, RMSD, ecuación 3). RMSD se da por la siguiente fórmula:

Ecuación 3. Fórmula general del RMSD

$$RMSD(a,b) = \sqrt{\frac{\sum_{i=1}^{n} |r_{ai} - r_{bi}|^2}{n}}$$

Donde r_{ai} y r_{bi} son las posiciones del átomo i en la estructura a y la estructura b, respectivamente, y donde las estructuras a y b se han superpuesto óptimamente. El ajuste se realizó utilizando el algoritmo McLachlan (McLachlan, 1982). RMSD mide la similitud de las posiciones atómicas.

RMSD es uno de los instrumentos más utilizados para la comparación de estructuras. Sin embargo, el RMSD tiene algunos aspectos negativos: la mejor alineación no siempre significa un RMSD mínimo. La importancia de la RMSD depende del tamaño de las estructuras, además varía con el tipo de proteína. No es una buena medida cuando no se pueden superponer simultáneamente todas las partes equivalentes de las proteínas.

4. Técnicas de optimización: Técnicas metaheurísticas e inteligencia de enjambre

Ahora se hablará sobre las diferentes técnicas de optimización, la principal fuente de información para este apartado proviene del documento (González, 2013).

La palabra “heurística” tiene su origen en la antigua palabra griega “heuriskein”, el arte del descubrimiento de nuevas estrategias (reglas) para resolver problemas. El sufijo “meta”, también griego, significa “metodología de alto nivel”. El término “metaheurística” fue introducido por F. Glover en el artículo (Glover, 1986).

Los métodos de búsqueda metaheurística son definidos pues como metodologías (plantillas) generales de alto nivel que pueden ser utilizadas como estrategias en el diseño de heurísticas subyacentes que resuelvan problemas de optimización específicos. Las metaheurísticas permiten abordar instancias de problemas complejos obteniéndose soluciones satisfactorias en un tiempo razonable, pero sin garantizar el descubrimiento de soluciones óptimas globales. Las metaheurísticas han recibido más y más popularidad en los últimos 20 años. Su uso en diferentes aplicaciones demuestra su eficiencia y efectividad en la resolución de problemas complejos.

A la hora de diseñar metaheurísticas se pueden tener en cuenta dos criterios contradictorios: la exploración del espacio de búsqueda (diversificación) y la explotación de las mejores soluciones encontradas (intensificación). Las regiones prometedoras se determinan a través de las “buenas” soluciones encontradas.

En la intensificación, las regiones prometedoras son exploradas más profundamente con la esperanza de encontrar mejores soluciones. Y, por el contrario, en la diversificación, las regiones sin explorar deben visitarse para asegurarnos de abordar todas las posibles regiones del espacio de búsqueda y así, asegurarnos de que la búsqueda no está limitada solamente a una porción del mismo.

Existen otros muchos criterios de clasificación para las metaheurísticas, algunos ejemplos son:

- Basadas en la naturaleza o no basadas en la naturaleza: Muchas metaheurísticas están inspiradas en procesos naturales. Algunos ejemplos son los algoritmos evolutivos y los sistemas inmunológicos artificiales (*artificial immune systems*) en la biología; hormigas (*ants*), colonias de abejas (*bees colonies*) y la optimización de enjambre de partículas (*Particle Swarm Optimization, PSO*) de la inteligencia colectiva (*swarm intelligence*) de diferentes especies; y el enfriamiento simulado (*simulated annealing*) de la física.

- Las que utilizan memoria y las que no: Algunas metaheurísticas no tienen memoria, es decir, no utilizan información extraída dinámicamente durante el proceso de búsqueda. Por otro lado, otras metaheurísticas utilizan memoria que contiene información extraída durante el proceso de búsqueda.

- Deterministas versus estocásticas: Una metaheurística determinista resuelve un problema de optimización tomando decisiones deterministas. Las metaheurísticas estocásticas aplican reglas aleatorias durante la búsqueda. Si considerásemos la misma solución inicial en los algoritmos deterministas siempre obtendríamos la misma solución final. Por contra, los algoritmos estocásticos podrían obtener diferentes soluciones finales con la misma solución inicial. Esta característica debe tenerse en cuenta en la evaluación del rendimiento de los algoritmos metaheurísticos.

- Basadas en población o en una única solución (trayectoria): Los algoritmos basados en una única solución manipulan y transforman una única solución durante el proceso de búsqueda mientras que, los algoritmos basados en población (por ejemplo, PSO o los algoritmos evolutivos), manipulan una población entera de soluciones. Estas dos familias poseen características

complementarias: las metaheurísticas basadas en una única solución están orientadas a la explotación ya que presentan la capacidad de intensificar las búsquedas en determinadas regiones. Por su parte, las metaheurísticas basadas en población están orientadas a la exploración al permitir una mayor diversificación por el espacio de búsqueda. En los siguientes apartados adoptaremos esta clasificación para organizar los algoritmos. Como ya veremos, los algoritmos de ambas familias comparten muchos mecanismos de búsqueda.

- Iterativos versus constructivos: En los algoritmos iterativos comenzamos con una solución completa (o una población de soluciones) y la transformamos en cada iteración utilizando varios operadores de búsqueda. Los algoritmos constructivos comienzan con una solución vacía y en cada etapa se asigna el valor a una variable de decisión hasta completarla. La mayoría de las metaheurísticas son algoritmos iterativos.

A continuación, profundizaremos un poco más sobre un tipo de metaheurística. Como ya hemos explicado anteriormente, existen muchas posibles clasificaciones para organizar las distintas metaheurísticas. Sin embargo, nosotros nos hemos centrado en la que las organiza teniendo en cuenta el número de soluciones que optimizan, concretamente basadas en una población de soluciones. Debido a que este tipo de metaheurística es la que más representa al algoritmo que se trata en este trabajo.

Las metaheurísticas basadas en población (P-metaheurísticas) pueden ser vistas como una mejora iterativa aplicada sobre una población de soluciones. Primero se inicializa la población correspondiente para, a continuación, generar una nueva población de soluciones. Finalmente, esta nueva población se combina con la inicial aplicando algunas funciones de selección. El proceso de búsqueda en estas técnicas se detiene cuando la condición de finalización se cumple.

Técnicas como los algoritmos evolutivos (*Evolutionary Algorithms*, EAs), algoritmos genéticos (*Genetic Algorithms*, GAs), evolución diferencial (*Differential Evolution*, DE) o la optimización de enjambre de partículas (PSO) pertenecen a esta clase de metaheurísticas.

Haciendo distinción de fases en esta clase de algoritmos (ilustración 9), podríamos obtener una fase de generación y otra de reemplazo, en la primera fase una nueva población de soluciones es creada y en la segunda se aplican las correspondientes funciones de selección de soluciones. Estas dos fases pueden realizarse sin utilizar memoria. En este caso, se aplican únicamente sobre la población actual. Por otro lado, de considerarse un sistema con memoria, podríamos almacenar un histórico de búsquedas que nos proporcionase información útil para optimizar la búsqueda.

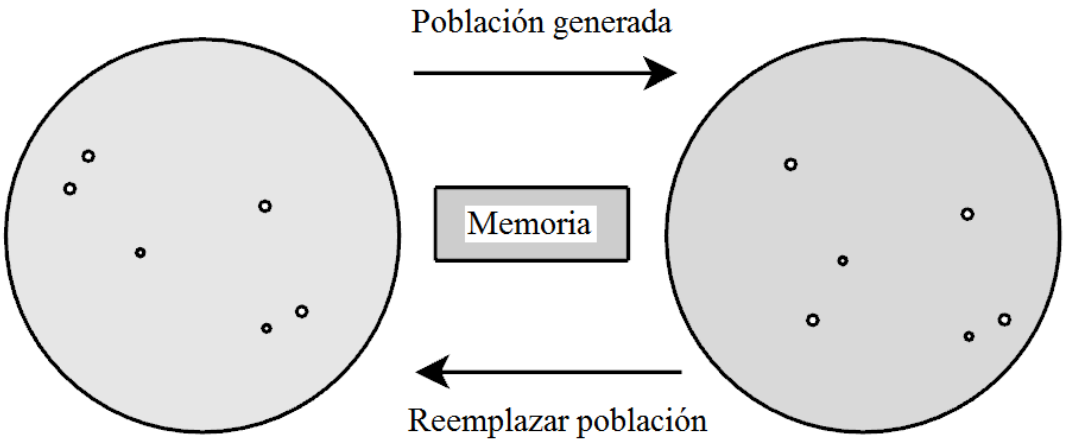

Ilustración 9. Principios de las metaheuríticas basadas en población. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

4.1. Ejemplos de técnicas basadas en población

- Los Algoritmos Genéticos (GA) fueros desarrollados para comprender los procesos adaptativos de los sistemas naturales. Tiempo después, los mismos autores aplicaron estos conocimientos en optimización y aprendizaje automático (*machine learning*) en los 80. Los GAs son un tipo muy popular de algoritmos evolutivos que, tradicionalmente, están asociados con el uso de una representación binaria. Sin embargo, actualmente pueden encontrarse numerosos trabajos que aplican otro tipo de representaciones. Normalmente un GA aplica un operador de cruce sobre dos

soluciones y un operador de mutación que modifica aleatoriamente el contenido del individuo correspondiente para proporcionar diversidad a la búsqueda. Los GAs utilizan selección probabilística que básicamente es una selección basada en proporciones. El reemplazo (selección de supervivientes) es generacional, es decir, los padres son reemplazados sistemáticamente por los hijos. El operador de cruce puede estar basado en 'n' puntos o ser uniforme y la mutación se realiza a nivel de bits.

- La Evolución Diferencial (DE) es una de las propuestas que más éxito ha cosechado en la optimización de problemas continuos. La idea principal bajo la que se apoya el algoritmo DE es la utilización de las diferencias existentes entre los diferentes vectores de la población para perturbar al vector población. Esta idea se integró en nuevos operadores de recombinación de dos o más soluciones y en nuevos operadores de mutación para dirigir las búsquedas hacia "buenas" soluciones. La comunidad DE es relativamente nueva y sigue creciendo.

- El Algoritmo de Enseñanza-Aprendizaje (*Teaching-Learning-Based Optimization* - TLBO) es una técnica de optimización novedosa. Es una P-metaheurística que define una población de soluciones que evoluciona conjuntamente. Esta población simula ser una clase de alumnos. Del mismo modo que en cualquier algoritmo evolutivo, el algoritmo TLBO define un conjunto de cromosomas (variables de decisión) que deben ser optimizadas. A su vez, cada cromosoma se corresponde con una asignatura cursada por cada alumno (individuo). Por todo ello, cuanto mejores sean las calificaciones obtenidas por los alumnos en las asignaturas cursadas, mejor será su aprendizaje y, por tanto, su calidad o *fitness*.

El algoritmo TLBO distingue dos tipos de aprendizaje: el aprendizaje obtenido por el profesor y el obtenido a través de los compañeros. De este modo, el primer aprendizaje se realiza tomando como referencia los conocimientos del profesor (mejor individuo de la población) en cada asignatura (variable de decisión), y el segundo aprendizaje se realiza tomando como referencia los conocimientos de la clase (nota media de la clase en cada asignatura).

4.2. Inteligencia de enjambre (*Swarm Intelligence*)

Los algoritmos inspirados en el comportamiento colectivo de organismos como las hormigas, abejas, avispas, peces y/o pájaros son conocidos como algoritmos basados en inteligencia colectiva (*Swarm Intelligence*). La inteligencia colectiva está basada en los comportamientos sociales presentados por organismos que compiten, generalmente, por la obtención de alimentos.

Las principales características de estos algoritmos radican en la definición de agentes poco sofisticados que cooperan mediante comunicaciones indirectas para explorar el espacio de decisión definido por el correspondiente problema de optimización. Algunos de los algoritmos más conocidos de esta clase son: la Optimización por Colonia de Hormigas (*Ant Colony Optimization* - ACO), la Optimización de Enjambre de Partículas (PSO) y la Colonia Artificial de Abejas (*Artificial Bee Colony* - ABC):

- La idea básica de la Optimización por Colonia de Hormigas (ACO) es imitar el comportamiento cooperativo de las hormigas para resolver un problema de optimización (ilustración 10). Esta metaheurística puede ser vista como un sistema multi-agente donde cada uno de estos agentes está, a su vez, inspirado por el comportamiento de una hormiga. El algoritmo ACO ha sido aplicado para resolver problemas de optimización combinatoria y ha logrado buenos resultados en una gran cantidad de ellos (por ejemplo, problemas de planificación, de enrutado o de asignación).

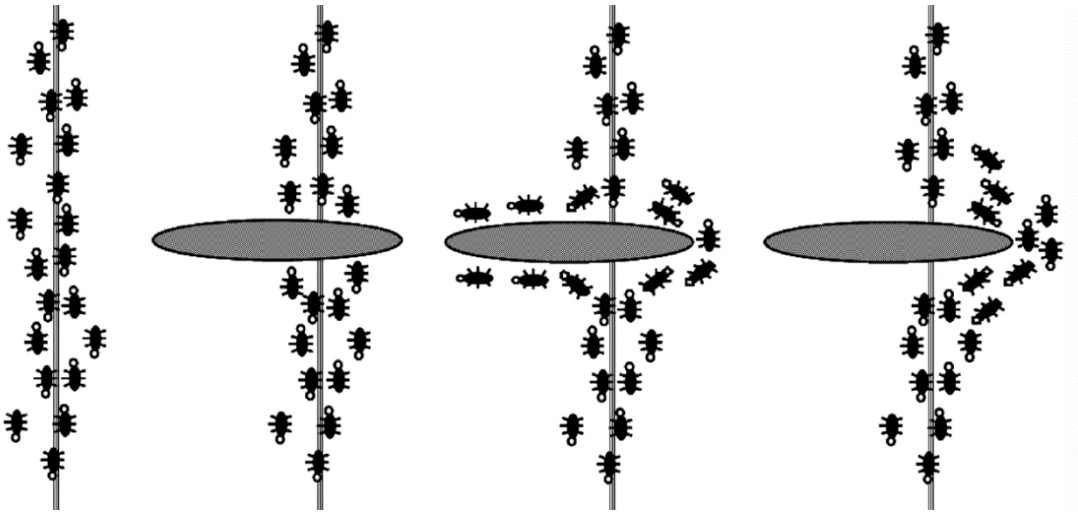

Ilustración 10. Comportamiento de una colonia de hormigas cuando un obstáculo aparece de repente. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

Algunos aspectos interesantes del comportamiento de las hormigas es que simples hormigas, siguiendo un comportamiento colectivo, son capaces de realizar tareas complejas como transportar alimento o encontrar las rutas más cortas entre las fuentes de alimento y sus colonias. Los algoritmos ACO definen mecanismos de comunicación simples y una hormiga es capaz de encontrar la mejor ruta entre dos puntos.

- La Optimización de Enjambre de Partículas (PSO) es otra metaheurística basada en población inspirada en el comportamiento colectivo. Este algoritmo imita el comportamiento social de organismos naturales como los pájaros o los peces a la hora de encontrar un lugar con el alimento suficiente. Inicialmente, el algoritmo PSO fue diseñado para optimizar problemas continuos.

- La Colonia Artificial de Abejas (ABC) es un algoritmo reciente propuesto por D. Karaboga en 2005 (Karaboga, 2005) e inspirado por el comportamiento de las abejas de la miel. El algoritmo ABC es un algoritmo basado en población donde las soluciones (denominadas fuentes de alimento) son explotadas por diferentes agentes (distintas clases de abejas). De este modo, el objetivo principal del algoritmo es descubrir las fuentes de alimento con una mayor cantidad de néctar, la cual

representa la calidad de la solución. En el algoritmo ABC las abejas vuelan alrededor de un espacio de búsqueda multidimensional.

Estas abejas presentan diferentes comportamientos, mientras que algunas (abejas obreras y observadoras) escogen fuentes de alimento guiándose por su experiencia y la de la colmena, otras (abejas exploradoras) escogen fuentes de alimento al azar. Al final el algoritmo irá almacenando las soluciones que presentan una mayor cantidad de néctar. De este modo, el algoritmo ABC combina procesos de búsqueda local (obreras y observadoras) con procesos de búsqueda global (exploradoras) tratando de balancear las propiedades de explotación y exploración.

En el algoritmo ABC la colonia de abejas está compuesta por tres clases de abejas: las obreras, las observadoras y las exploradoras. Aquellas que esperan en la colmena información relevante para explotar una u otra fuente son las observadoras y, las encargadas de proporcionar esta información y de explotar el néctar, son las obreras. Por otro lado, las abejas que realizan búsquedas aleatorias son las exploradoras.

En el algoritmo ABC, la colonia está formada únicamente por dos clases de abejas, la primera mitad la componen abejas obreras y la segunda mitad abejas observadoras. Y dado que cada fuente de alimento es únicamente explotada por una abeja obrera, el número de fuentes de alimento explotadas se corresponde con el número de abejas de la colmena. Cuando las fuentes explotadas se agotan, son las abejas exploradoras las encargadas de encontrar una nueva fuente que la sustituya.

- El Algoritmo de las Luciérnagas (*Firefly Algorithm* - FA) es uno de los últimos algoritmos basados en la naturaleza que se han propuesto. Este algoritmo fue definido por X-S. Yang en el artículo (Yang, 2009) y está inspirado en el comportamiento de las luciérnagas de la luz. Estos insectos pertenecen a una familia de coleópteros polífagos caracterizados por su capacidad de emitir luz (bioluminiscencia). Existen muchas especies y normalmente viven en pantanos o en áreas húmedas y boscosas donde sus larvas pueden alimentarse. Las luciérnagas

hembra utilizan su capacidad bioluminiscente para atraer a los machos que vuelan en los alrededores. En este algoritmo se suelen asociar estos patrones de luz con los objetivos que deben ser optimizados.

X-S. Yang formuló este algoritmo teniendo en cuenta las siguientes cuestiones:

- Todas las luciérnagas son asexuales por lo que todas se ven atraídas por todas.
- La atracción es proporcional a su brillo y, dadas dos luciérnagas, la menos brillante se ve atraída por (se mueve hacia) la más brillante; sin embargo, el brillo puede disminuir a medida que la distancia aumenta.
- Si no existen luciérnagas más brillantes que una luciérnaga dada, ésta se mueve aleatoriamente.

- El Algoritmo de Búsqueda Gravitacional (*Gravitational Search Algorithm* - GSA) es una metaheurística reciente y está basada en la ley gravitatoria y la consecuente interacción entre masas. En física, la interacción gravitatoria es una de las cuatro interacciones fundamentales, o lo que es lo mismo, uno de los cuatro tipos de campos cuánticos mediante los cuales interactúan las partículas (las otras son la interacción electromagnética, la interacción nuclear débil y la interacción nuclear fuerte). Cada partícula en el universo se ve atraída por otra. Consecuentemente, la gravedad actúa en todas partes y esto la diferencia de otras fuerzas presentes en la naturaleza.

El algoritmo GSA está inspirado en las teorías físicas de Newton. Este algoritmo define una población compuesta por diferentes agentes que simulan ser un conjunto de masas. Mediante la fuerza gravitatoria, cada agente del sistema puede ver la posición de los demás. Entonces, la fuerza gravitatoria es utilizada para transferir información entre los diferentes agentes. A su vez, estos agentes son considerados objetos y su calidad viene representada por sus masas.

Todos estos objetos se atraen unos a otros provocando un movimiento global de todos los objetos hacia aquellos más pesados (de mejor calidad). La posición de cada agente se corresponde con una posible solución del problema de optimización y su masa determina su calidad o *fitness*. De este modo, la población del algoritmo puede verse como un pequeño sistema de masas que obedecen las leyes gravitacionales y de movimiento de Newton.

- El Algoritmo de las Ranas Saltarinas (*Shuffled Frog Leaping Algorithm* - SFLA) es otro algoritmo evolutivo basado en inteligencia colectiva diseñado para abordar problemas de optimización combinatoria. Este algoritmo está inspirado en la evolución de los memes a través de la interacción entre individuos y de un intercambio global de información.

El algoritmo SFLA fue diseñado teniendo en cuenta las ventajas que proporcionan los algoritmos meméticos (*memetic algorithm*, MA) y los algoritmos basados en comportamientos sociales como, por ejemplo, el PSO. El concepto memético surge del término meme, el cual simplemente define una unidad de información intelectual o cultural que sobrevive lo suficiente como para ser reconocido como tal y que puede evolucionar a lo largo de generaciones.

Más concretamente, el algoritmo SFLA simula una evolución memética de un conjunto de ranas (soluciones). En este algoritmo las ranas individuales no son importantes ya que son consideradas únicamente como fuentes de información para los diferentes memes. En el SFLA, la población está formada por un conjunto de ranas (soluciones) que se organizan en varios subconjuntos (o memes/memeplexes), los cuales simulan diferentes “charcos”.

En cada memeplex las ranas evolucionan teniendo en cuenta la riqueza genética de los individuos que la componen. Tras un determinado número de pasos evolutivos, algunas ranas son cambiadas de subpoblación para tratar de enriquecer la calidad genética de las ranas de la subpoblación destino. Los procesos de evolución e

intercambio se repiten hasta alcanzar la condición de finalización y son los que caracterizan a este algoritmo.

5. Optimización multiobjetivo

Al igual que el apartado anterior, la fuente primordial de donde se ha extraído la información para este apartado también es el documento de (González, 2013).

En este apartado incluimos la información necesaria para comprender todos los aspectos multiobjetivo que se han utilizado o aplicado a este trabajo final de grado. Primero, introduciremos el mundo de la optimización multiobjetivo, proporcionando información sobre sus orígenes, destacando las diferencias existentes entre esta clase de optimización y la optimización mono-objetivo. A continuación, explicamos los conceptos más importantes de la optimización multiobjetivo, describiendo el significado de, por ejemplo, el concepto de dominancia, los frentes de Pareto, entre otros muchos. Una vez concluidas las explicaciones matemáticas, incluimos numerosos campos donde se aplican o se han aplicado con éxito metaheurísticas multiobjetivo.

5.1. Introducción a la optimización multiobjetivo

Muchos dominios están prestando una creciente atención a la optimización de problemas multiobjetivo complejos. En realidad, los problemas de optimización prácticos rara vez optimizan una única función objetivo. Por lo general, suelen manejarse varios objetivos en conflicto. Por ejemplo, a la hora de diseñar un determinado producto, uno debe minimizar su coste y maximizar su calidad. Diversas aéreas de investigación (por ejemplo, telecomunicaciones, ingeniería, bioinformática, logística, transporte, aeronáutica, economía...) están al corriente de la creciente importancia de los problemas de optimización multiobjetivo (*Multiobjective Optimization Problem* - MOP).

La optimización multiobjetivo tiene sus orígenes en el siglo diecinueve a través de los trabajos de F. Edgeworth y V. Pareto en el campo de la economía. Las ideas transmitidas por estos autores fueron ampliamente utilizadas en ciencias económicas durante varias décadas y después fueron también entrando en campos como la ingeniería. Actualmente, la optimización multiobjetivo es uno de los temas más importantes en ciencia e ingeniería.

La complejidad de esta clase de problemas (MOPs) está directamente relacionada con el tamaño del problema que queremos resolver, es decir, el número de objetivos definidos, el tamaño del espacio de búsqueda... pero, a su vez, el tiempo de búsqueda requerido para resolverlos debe ser razonable para que su aplicación sea de verdad práctica. Por todo esto, avanzar en el desarrollo de metaheurísticas multiobjetivo ha sido ampliamente investigado desde finales de los 80.

La solución óptima en un MOP no suele ser una única solución como en los problemas de optimización mono-objetivo, sino un conjunto de soluciones conocidas como soluciones Pareto óptimas. Una solución es Pareto óptima si no es posible mejorar el valor de un objetivo dado sin perjudicar alguno de los demás. Este conjunto de soluciones representa soluciones de compromiso entre los diferentes objetivos en conflicto.

El principal objetivo que debemos alcanzar cuando resolvamos un problema multiobjetivo es obtener este conjunto de soluciones Pareto óptimas y, consecuentemente, el frente de Pareto (la representación gráfica de este conjunto de soluciones). A pesar de todo esto, cuando aplicamos metaheurísticas, el objetivo ha de ser obtener una buena aproximación al conjunto de soluciones Pareto óptimas teniendo en cuenta dos factores: la convergencia hacia el frente de Pareto óptimo y la obtención de soluciones con una dispersión uniforme. La primera propiedad asegura la generación de soluciones Pareto cuasi-óptimas, mientras que la segunda indica la buena distribución de las soluciones encontradas. Si comparamos la optimización multiobjetivo con la mono-objetivo, su dificultad radica en los siguientes hechos:

- No existen definiciones comúnmente utilizadas sobre la optimalidad global de una solución como en la optimización mono-objetivo. La relación de orden entre las soluciones de un MOP es sólo parcial y la elección final depende del tomador de decisiones.

- El número de soluciones Pareto óptimas aumenta de acuerdo al tamaño del problema y, principalmente, con el número de objetivos considerados. Teniendo en cuenta este factor, el número de soluciones Pareto óptimas puede verse incrementado de forma exponencial con respecto al tamaño del problema.

- La estructura del frente de Pareto, es decir, su continuidad, forma (cóncava, convexa...), modalidad... depende del MOP estudiado. Por ejemplo, las soluciones Pareto óptimas pueden estar perfectamente localizadas en la frontera e interior de una función convexa envolvente con soluciones factibles.

5.2. Conceptos importantes

Ahora explicaremos los conceptos más importantes de la optimización multiobjetivo considerando para estas definiciones la minimización de todos los objetivos.

Definición: Problema de optimización multiobjetivo (ecuación 4). Un problema de optimización multiobjetivo puede definirse como:

Ecuación 4. Definición general de un problema de optimización multiobjetivo

$$MOP = \{\min F(x) = (f_1(x), f_2(x), \dots, f_n(x)) \; s.c. \; x \in S\}$$

Donde n (n ≥ 2) es el número de objetivos, $x = (x_1, \dots, x_k)$ es el vector que representa las variables de decisión y S representa el conjunto de soluciones factibles. $F(x) = f_1(x), f_2(x), \dots, f_n(x))$ es el vector de objetivos a ser optimizado.

El espacio de búsqueda S representa el espacio de decisión del MOP. El espacio al que pertenece el vector objetivo es denominado espacio objetivo. El vector F puede ser definido como las funciones de coste del espacio de decisión en el espacio objetivo que evalúan la calidad de cada solución $(x_1, \dots, x_k)$ mediante la asignación

de un vector objetivo, el cual representa la calidad de la solución (o *fitness*). El conjunto Y = F(S) representa los puntos factibles en el espacio objetivo y $y = F(x) = (y_1, y_2, \dots, y_n)$, donde $y_i = f_i(x)$, es un punto en el espacio objetivo, véase en la ilustración 11.

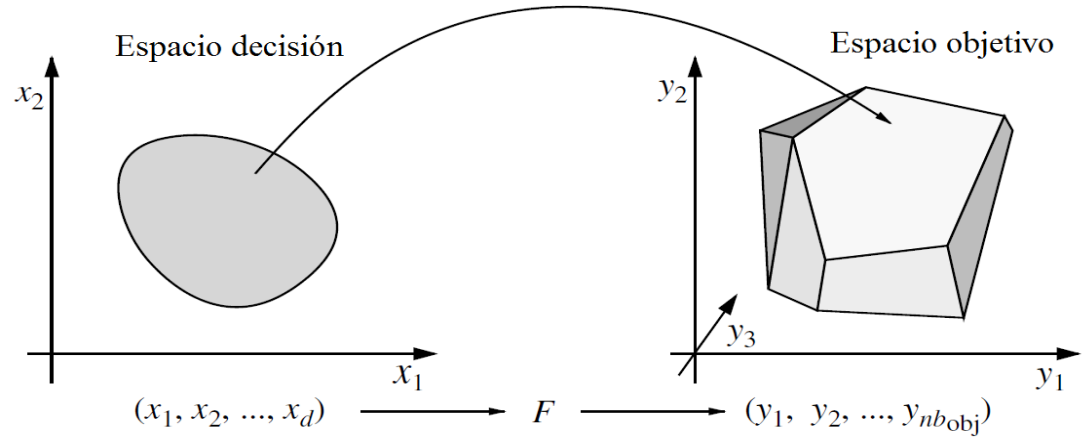

Ilustración 11. Espacio de decisión y espacio objetivo de un MOP. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

No es usual encontrar una solución x^* asociada a un determinado vector de decisión (ecuación 5) que optimice todos los objetivos:

Ecuación 5. Fórmula de una x perteneciente a un determinado vector de decisión.

$$\forall x \in S, f_i(x^*) \leq f_i(x), i = 1, 2, \dots, n$$

Dado que esta situación no suele presentarse en los MOPs de la vida real, donde normalmente encontramos varios objetivos en conflicto, otros conceptos deben formularse para poder considerar la optimilidad de las soluciones. Una relación de orden parcial entre soluciones puede definirse como relación de dominancia.

Definición: Dominancia Pareto (ecuación 6). Un vector objetivo $u = (u_1, \dots, u_n)$ se dice que domina a otro vector $v = (v_1, \dots, v_n)$ (denotado por $u \prec v$) si y sólo si ningún componente de v es menor que el correspondiente componente de u, siendo al menos un componente de u estrictamente menor, es decir:

Ecuación 6. Comprobación de dominancia de Pareto.

$$\forall i \in \{1, \dots, n\} : u_i \leq v_i \wedge \exists i \in \{1, \dots, n\} : u_i < v_i$$

El concepto utilizado generalmente es el de optimalidad de Pareto. La definición de la optimalidad de Pareto proviene directamente del concepto de dominancia. Este concepto fue propuesto inicialmente por F. Edgeworth en 1881 y extendido por V. Pareto en 1896. Una solución Pareto óptima indica que es imposible encontrar una solución que mejore su calidad en algún criterio sin, a su vez, penalizar la calidad de otro criterio.

Definición: Optimalidad Pareto. Una solución $x^* \in S$ es Pareto óptima si para todas las $x \in S, F(x)$ no domina a $F(x^*)$, lo que significaría está en la ecuación 7.

Ecuación 7. Fórmula de optimalidad de Pareto.

$$F(x) \prec F(x^*)$$

Gráficamente, una solución x^* es Pareto óptima si no existe otra solución x tal que el punto F(x) esté presente en el interior de su cono de dominancia a $F(x^*)$, representado por la superficie definida por las proyecciones de F(x). Por lo general, en los problemas de optimización mono-objetivo encontramos una única solución global, pero en los MOP podemos tener un conjunto de soluciones conocidas como conjunto Pareto óptimo. La imagen de este conjunto en el espacio objetivo es conocida como frente de Pareto, se puede observar en la ilustración 12.

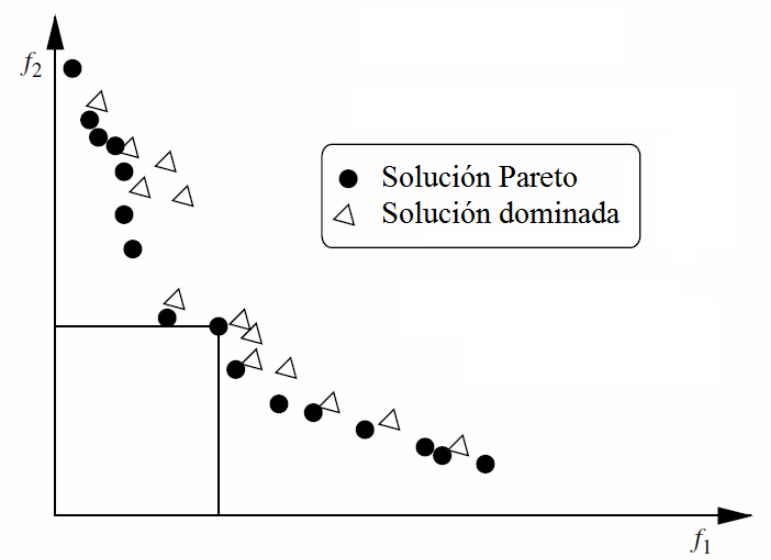

Ilustración 12. Soluciones no-dominadas en el espacio objetivo. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

Definición: Conjunto Pareto óptimo (ecuación 8). Para un determinado MOP (F, S), el conjunto Pareto óptimo se define como sigue:

Ecuación 8. Fórmula de un conjunto de Pareto para un determinado problema de optimización multiobjetivo.

$$P^* = \{x \in \mathrm{S} / \nexists x' \in \mathrm{S}, \mathrm{F}(x') \prec F(x)\}$$

Definición: Frente de Pareto. Para un determinado MOP (F, S) y su correspondiente conjunto Pareto óptimo P^*, el frente de Pareto se define en la ecuación 9.

Ecuación 9. Fórmula del frente de Pareto.

$$PF^* = \{F(x), x \in P^*\}$$

El frente de Pareto se corresponde con el conjunto Pareto óptimo en el espacio objetivo. Obtener el frente de Pareto de un MOP es el principal objetivo de la optimización multiobjetivo. Sin embargo, dado que un frente de Pareto puede contener un gran número de puntos, una buena aproximación a este frente que contenga un limitado número de soluciones Pareto puede sernos también de utilidad.

Estas soluciones deben estar lo más próximas posible al frente de Pareto exacto o verdadero y, además, deben presentar una dispersión uniforme. Si alguna de estas dos propiedades no se cumple, la aproximación obtenida podría no ser de utilidad al tomador de decisiones que debe poseer una información completa sobre el frente de Pareto.

Si nos fijamos, por ejemplo, en los tres frentes mostrados en la ilustración 13. La primera grafica (izquierda) muestra unas soluciones con una buena dispersión, pero muy alejadas del frente de Pareto verdadero (línea continua); ese frente no sería de utilidad porque no proporciona ninguna solución Pareto óptima. La segunda gráfica (centro) contiene un conjunto de soluciones muy cercanas al frente de Pareto verdadero pero algunas regiones del frente de Pareto verdadero no son cubiertas por las soluciones encontradas (mala distribución). Por ello, el tomador de decisiones podría no disponer de información importante sobre el frente de Pareto y proporcionar así una solución que no sea de utilidad. Finalmente, el último frente (derecha) cumple ambas propiedades y presenta una buena convergencia y dispersión.

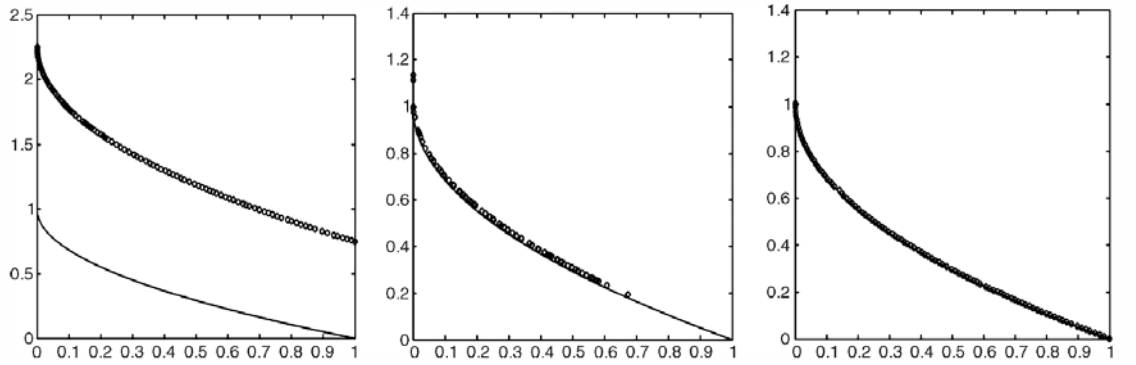

Ilustración 13. Ejemplo de frentes de Pareto. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

Definición: Vector ideal. Un punto $y^* = (y^*_1, \dots, y^*_2, \dots, y^*_n)$ es un vector ideal si minimiza cada función objetivo f_i en F(x), como lo define la ecuación 10.

Ecuación 10. Fórmula general de un vector ideal si minimiza la función objetivo.

$$y^*_i = \min\big(f_i(x)\big), x \in S, i \in [1, n]$$

El vector ideal es generalmente una solución utópica en el sentido de que no suele ser una solución factible en el espacio de decisión. Sin embargo, algunos tomadores de decisiones definen un vector de referencia indicando el valor deseado en cada función objetivo.

Esto generaliza el concepto de vector ideal. De este modo, el tomador de decisiones puede especificar ciertos niveles de aspiración $\bar{z_i}, i \in [1, n]$ a alcanzar en cada función f_i. Los niveles de aspiración indican los niveles de aceptación en el espacio objetivo. Una solución Pareto óptima debe satisfacer todos los niveles de aspiración y es denominada como una solución satisfactoria.

Definición: Punto extremo o *nadir*. Un punto $y^* = (y^*_1, \dots, y^*_2, \dots, y^*_n)$ es un punto extremo si maximiza toda función objetivo f_i de F en el conjunto Pareto, como se puede observar en la ecuación 11.

Ecuación 11. Fórmula del punto extremo si maximiza toda función objetivo del conjunto Pareto.

$$y^*_i = \max\big(f_i(x)\big), x \in P^*, i \in [1, n]$$

Los puntos extremos e ideales proporcionan mucha información sobre el rango de valores del frente de Pareto óptimo, véase en la ilustración 14.

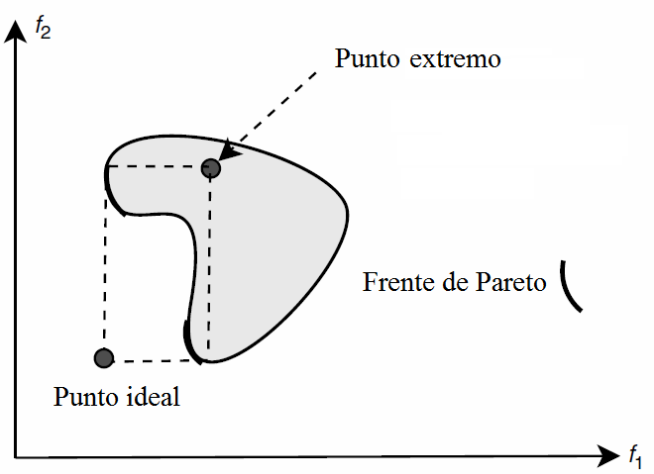

Ilustración 14. Punto extremo y punto ideal de un MOP. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

Definición: Función de utilidad. Una función (o valor) de utilidad v, la cual representa las preferencias del tomador de decisiones (ecuación 12), convierte el vector objetivo en una función escalar:

Ecuación 12. Fórmula de la función de utilidad de las preferencias del tomador de decisiones.

$$v : R^n \rightarrow R$$

La función de utilidad ha de ser minimizada para adaptarse a las preferencias del tomador de decisiones. En optimización multiobjetivo, el concepto de mínimo local puede ser generalizado a una solución Pareto óptima local.

Definición: Solución Pareto óptima local (ecuación 14). Una solución x es Pareto óptima local si y sólo si cumple la ecuación 13.

Ecuación 13. Condición de que x es una solución de Pareto óptima.

$$\forall w \in N(x), F(w) \text{ no domina a } F(x)$$

Donde $N(x)$ representa la vecindad de la solución x. Algunas soluciones Pareto óptimas pueden ser obtenidas a partir de la resolución de la siguiente expresión matemática:

Ecuación 14. Fórmula de una solución Pareto óptima local

$$(MOP_\lambda) = \left\{ \min F(x) = \sum_{i=1}^{n} \lambda_i f_i(x) \, s.c. \; x \in S \right\}$$

Con $\lambda_i \geq 0$ para $i = 1, \dots, n$ y $\sum_{i=1}^{n} \lambda_i = 1$.

Estas soluciones son conocidas como soluciones apoyadas o soportadas. Las soluciones apoyadas se generan a partir de la resolución de (MOP_λ) utilizando diferentes vectores de pesos λ. La complejidad de (MOP_λ) es la misma que la del problema de optimización mono-objetivo subyacente.

Si los problemas de optimización mono-objetivo subyacentes presentan una complejidad polinomial, será relativamente sencillo generar soluciones apoyadas. Sin embargo, existen otras soluciones Pareto óptimas que no pueden obtenerse resolviendo una (MOP_λ) matemática. En realidad, estas soluciones, conocidas como soluciones no apoyadas o no soportadas, son dominadas por combinaciones convexas de soluciones apoyadas, es decir, puntos de la forma convexa $Y = F(S)$ (véase en la ilustración 15).

Existen también otras definiciones de dominancia tales como el concepto de dominancia leve o débil, la dominancia estricta o la ϵ-dominancia.

Definición de dominancia leve. Un vector objetivo $u = (u_1, \dots, u_n)$ se dice que domina levemente a otro vector $v = (v_1, \dots, v_n)$ (denotado por $u \preccurlyeq v$) si todos los componentes de u son menores o iguales que los correspondientes componentes de v, es decir, $\forall i \in \{1, \dots, n\}, u_i \leq v_i$ (véase en la ilustración 15).

Definición de dominancia estricta. Un vector objetivo $u = (u_1, \dots, u_n)$ se dice que domina estrictamente a otro vector $v = (v_1, \dots, v_n)$ (denotado por $u \prec\prec v$) si todos los componentes de u son menores que los correspondientes componentes de v, es decir, $\forall i \in \{1, \dots, n\}, u_i < v_i$ (véase en la ilustración 15).

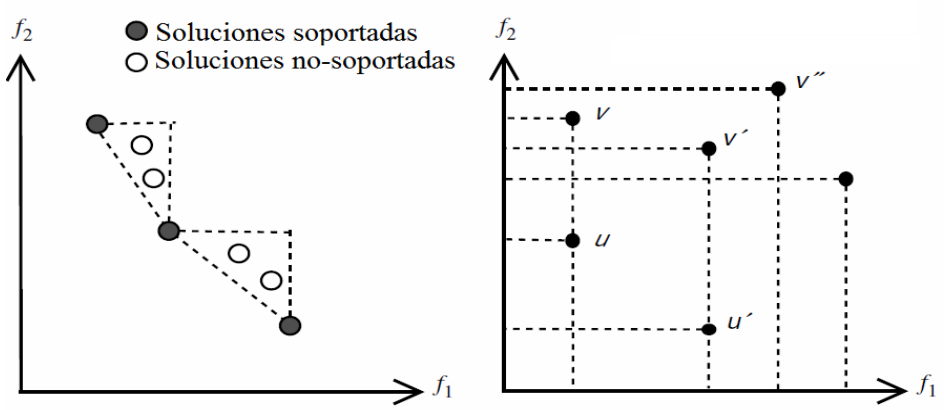

Ilustración 15. A la izquierda, soluciones apoyadas o soportadas y no-apoyadas o no-soportadas de un MOP. A la derecha, conceptos de dominancia leve y estricta. La solución u domina levemente a la solución v; la solución u domina estrictamente a las soluciones v' y v''. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

Definición de $\epsilon - dominancia$ (ilustración 16). Un vector objetivo $u = (u_1, \dots, u_n)$ se dice que ϵ-domina a otro vector $v = (v_1, \dots, v_n)$ (denotado por $u \prec_\epsilon v$) si y sólo si ningún componente de v es menor que el correspondiente componente de u$-\epsilon$ y al menos un componente de u$-\epsilon$ es estrictamente mejor, es decir, $\forall i \in \{1, \dots, n\}: u_i - \epsilon_i \leq v_i \wedge \exists i \in \{1, \dots, n\} : u_i - \epsilon_i < v_i$.

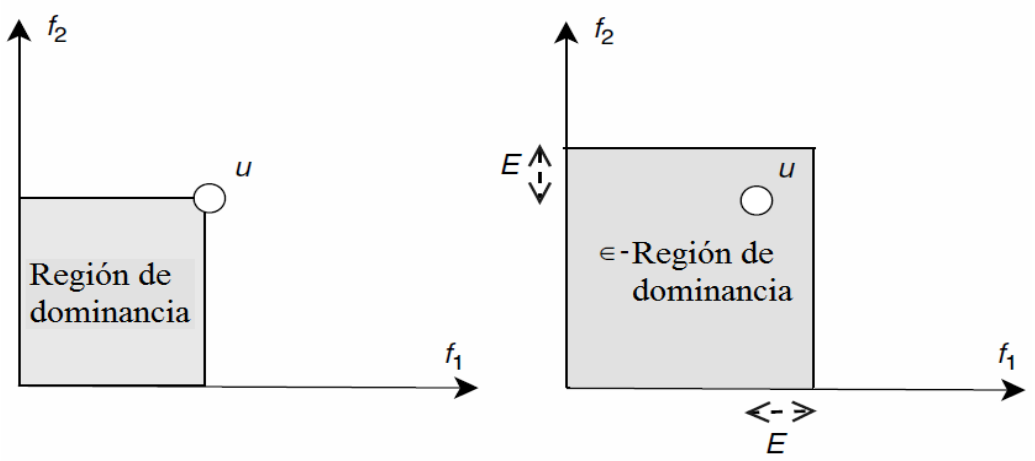

Ilustración 16. Concepto de $\epsilon-dominancia$. Esta imagen está sacada de David L. González Álvarez. Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Tesis Doctoral. Universidad de Extremadura. 2013

5.3. Aplicaciones reales de la optimización multiobjetivo

Un gran número de trabajos que tratan con MOPs se dedican a resolver problemas presentes en el mundo real. Dos aspectos clave son los principales causantes de este creciente interés.

El primero de ellos está relacionado con la gran cantidad de problemas del mundo real que requieren la optimización simultánea de diferentes funciones objetivo en conflicto.

Por otro lado, son muchas las nuevas metaheurísticas eficientes que han sido desarrolladas en los últimos años (por ejemplo, los algoritmos evolutivos multiobjetivo).

En realidad, las metaheurísticas multiobjetivo se vienen aplicando para resolver problemas reales desde 1960. Además, muchas son las áreas donde se han venido aplicando:

- Ingeniería: En los pasados 20 años, el diseño de sistemas en ingeniería (por ejemplo, mecánica, aeronáutica, química) ha impulsado la investigación y el desarrollo de metaheurísticas multiobjetivo. De hecho, numerosos problemas de diseño han sido formulados como MOPs (entre ellos destacamos, por ejemplo, los relacionados con el diseño de alas de avión o motores de coches). La formulación multiobjetivo de los problemas de diseño ha cosechado un gran éxito en el dominio de la ingeniería.

- Medio ambiente y energía: En la literatura, uno de los primeros papeles que ha desempeñado el modelado de problemas de optimización multiobjetivo está relacionado con los problemas de ámbito ambiental o energético (por ejemplo, la gestión de distribución de aguas o la medición de la calidad del aire). Este dominio será aún más importante en los próximos años debido al deterioro que sufre nuestro planeta y a la falta de recursos energéticos como el agua o recursos no renovables como el petróleo.

- Telecomunicaciones: En las últimas décadas las telecomunicaciones ha sido uno de los dominios donde más éxito han cosechado las metaheurísticas multiobjetivo (por ejemplo, en el diseño de antenas, el diseño de redes móviles, el diseño de satélites espaciales o la asignación de frecuencias). Este dominio continuará siendo relevante los próximos años debido a la continua evolución que están sufriendo la tecnología de redes (redes de sensores, redes ad-hoc, redes cognitivas).

- Control: Las aplicaciones clásicas sobre diseño óptimo de controladores siguen siendo ampliamente abordadas en términos de optimización multiobjetivo.

- Biología computacional y bioinformática: Numerosos problemas clave de la biología computacional o la bioinformática puede formularse también como MOPs. La optimización multiobjetivo está en sus etapas iniciales en este

dominio y un gran número de posibles trabajos aún pueden mejorarse desde este punto de vista (por ejemplo, inferencia filogenética, alineamiento de secuencias, predicción de estructuras e identificación de proteínas). Una revisión de numerosos problemas biológicos abordados mediante optimización multiobjetivo puede encontrarse en la siguiente referencia (Handl et al., 2007).

- Transporte y logística: Actualmente, este dominio genera un gran número de aplicaciones para los MOP (por ejemplo, la gestión de contenedores, el diseño de sistemas de red de mallas o el trazado de autopistas).

6. Introducción al paralelismo

Este apartado se divide en dos secciones. En la primera explicaremos los conceptos fundamentales sobre el paralelismo. La segunda sección se centrará en la programación paralela mediante el uso de OpenMP.

6.1. Conceptos fundamentales

El paralelismo surge como necesidad debido a:

- Necesidad de potencia de cálculo, como, por ejemplo, procesos complejos en tiempo real, simulaciones, problemas hasta ahora no atacables pero resolubles, etc.
- Limitaciones a las posibilidades de la arquitectura de computadores clásica, esto se debe a la presencia de múltiples cuellos de botella y limitaciones físicas.

Por estos motivos, la computación paralela es inevitable que vaya surgiendo y cogiendo más fuerza, debido a las tendencias tecnológicas, arquitectónicas y actuales, como, por ejemplo, el rumbo que han tomado los microprocesadores actuales (basados en sistemas multinúcleo).

El objetivo del paralelismo es aumentar la aceleración (*speedup*), y este concepto va de la mano con la eficiencia. La aceleración significa cuantas veces más rápido se ejecuta un programa al contar con “n” CPUs en vez de una. La eficiencia se calcula dividiendo la aceleración por el número de CPUs usadas; de esta forma, permitiendo saber cómo de eficientemente se usan dichas CPUs. La aceleración y eficiencia (ecuación 15) son dos de las métricas más usadas para medir el rendimiento de las arquitecturas paralelas, la aceleración es una métrica absoluta, mientras la eficiencia es una métrica relativa (al número de CPUs usadas).

Ecuación 15. Fórmulas de la aceleración y eficiencia.

$$Aceleración = Sn = {T^{secuencial}}/{T^{paralelo}}$$

$$Eficiencia = En = {T^{secuencial}}/{n * T^{paralelo}}$$

Donde $T^{secuencial}$ es el tiempo cuando el programa se ejecuta en secuencial, y $T^{paralelo}$ es el tiempo cuando el programa se ejecuta en paralelo (con cierto número de CPUs).

En definitiva, el paralelismo es una forma de computación en la cual varias tareas/cálculos pueden ejecutarse de forma simultánea, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños, que son posteriormente solucionados en paralelo. Hay varios tipos diferentes de paralelismo: a nivel de bit, nivel de instrucción, de datos y de tarea. El paralelismo es empleado sobre todo para la computación de alto rendimiento.

6.2. Programación paralela con OpenMP

Ahora hablaremos de OpenMP (*Open Multi-Processing*, OMP). OMP es una interfaz de programación de aplicaciones (API) utilizada para programar aplicaciones paralelas en multiprocesadores y procesadores multinúcleo (que es el tipo de hardware que se usa en este trabajo fin de grado). Esta API está especificada para C/C++ y Fortran, y está compuesta por directivas del compilador, bibliotecas de funciones y variables de entorno.

El modelo de ejecución de OMP es *fork/join* (ilustración 17). El hilo maestro se divide en un número determinado de hilos esclavos que se ejecutan en paralelo, y ya el entorno de ejecución asignará uno o varios hilos esclavos a los distintos procesadores disponibles.

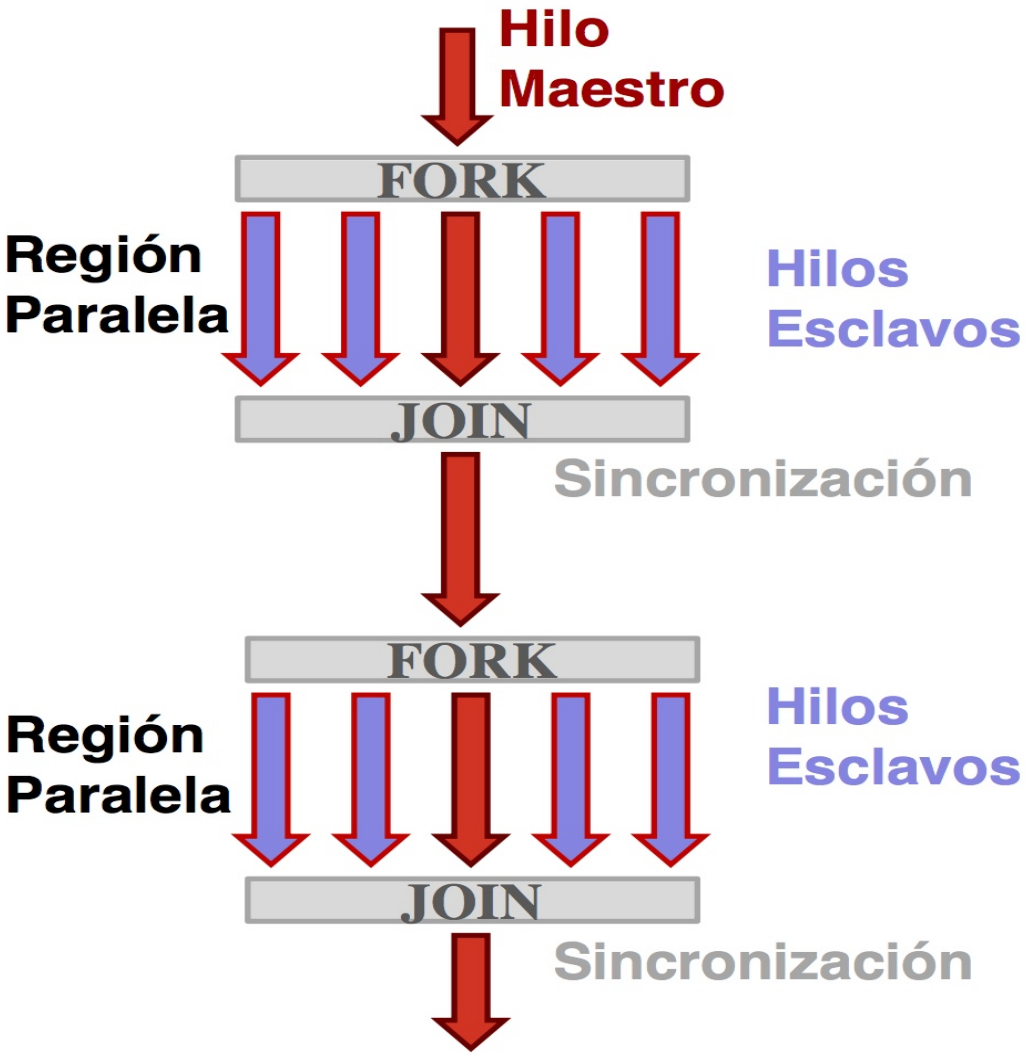

Ilustración 17. Modelo de ejecución de OpenMP. Esta imagen está sacada de la asignatura de Arquitectura de Computadores, Grado en Ingeniería Informática en Ingeniería de Computadores, de la Universidad de Extremadura.

Para poder aplicar OMP en un bucle hay que detectar si ese bucle tiene iteraciones dependientes o independientes de datos. Si son independientes son bucles paralelizables sino no.

Las distintas directivas de OMP siguen el siguiente formato general:

#pragma omp directiva [cláusula [cláusula]…]

Hay una gran variedad de directivas, pero tal vez las más populares sean las siguientes:

- Directiva for: define una región donde las iteraciones del bucle deben ejecutarse entre los hilos que llegan a esa región.

  #pragma omp for [cláusula [cláusula]…]

  bucle for

- Directiva parallel: define una región paralela, región que puede ser ejecutada por múltiples hilos en paralelo.

  #pragma omp parallel [cláusula [cláusula]…]

  bloque estructurado

- Directiva parallel for: define una región paralela que contiene un for. Es la combinación de las dos anteriores.

  #pragma omp parallel for [cláusula [cláusula]…]

  bucle for

También hay muchas cláusulas para controlar los datos durante la ejecución paralela, haciendo que estos datos puedan ser, por ejemplo, compartidos por todos los hilos o privados. A continuación, comentaremos algunas de ellas:

- Cláusula *firstprivate(lista)*: las variables de la lista son privadas para cada hilo, pero ese dato/s es inicializado según el valor antes de entrar en el bloque estructurado.

- Cláusula *schedule(dynamic)*: la cláusula *schedule* sirve para indicar el tipo de gestión y distribución entre los hilos de las iteraciones de los bucles paralelizados, es decir, la planificación que se seguirá. En concreto, con

schedule(dynamic), las iteraciones se asignan en bloques de q iteraciones a los hilos a medida que éstos lo solicitan. Por defecto q=1.

En general, las directivas vistas de OMP incluyen una barrera final implícita (sincronización), donde todos los hilos se esperan, a no ser que se use la cláusula *nowait*.

7. Hardware y software utilizados

En este apartado hablaremos de los recursos hardware que se han utilizado para la ejecución de nuestro algoritmo, además de cuáles son los recursos software externos de nuestro trabajo que también hemos usado.

7.1. Recursos hardware

Las ejecuciones se han realizado en un nodo del clúster de investigación del grupo ARCO. Este nodo tiene las siguientes características:

- Es un nodo multiprocesador con 4 procesadores del tipo: AMD Abu Dhabi 6376 2,3Ghz.
- Cada procesador es a su vez multicore, en concreto, cada procesador consta de 16 cores, lo que hace que el nodo en total tenga 64 cores.
- Además, tiene 12 módulos de memoria, con un total de 96 GB. La memoria de cada módulo es de este tipo: memoria de 8 GB DDR3 1866Mhz.

Por último, este nodo dispone de Linux como sistema operativo, en concreto una distribución de Ubuntu, la versión 14.04.

7.2. Recursos software

Los recursos software externos a nuestro trabajo que se han utilizado son: Tinker *Molecular Modeling Package* y ProFit. A continuación, comentaremos en qué consiste cada uno y los pasos necesarios para instalarlos.

7.2.1. Tinker Molecular Modeling Package

Tinker es un software que está compuesto de un paquete completo para mecánica y dinámica molecular. Es capaz de usar cualquier conjunto de parámetros, *Assisted Model Building with Energy Refinement* - AMBER (ff94, ff96, ff98, ff99, ff99SB), CHARMM (19, 22, 22/CMAP), entre otros. Además, permite su ejecución en sistemas operativos Windows, Unix-Linux y Mac, y su código fuente está disponible gratuitamente bajo una licencia restrictiva. El código está escrito en Fortran77 con algunas extensiones de C.

Para poder instalar el código fuente de Tinker en nuestro equipo, debemos instalar previamente los siguientes paquetes: *texinfo* y *gfortran*. Estos paquetes se instalan de la siguiente manera: *sudo apt-get install texinfo y sudo apt-get install gfortran*. Una vez instalados estos paquetes, nos descargamos el código fuente de Tinker de este enlace (https://dasher.wustl.edu/tinker/), nos dirigimos al directorio donde se encuentre el paquete descargado y hacemos los siguientes pasos:

```
tar -xzf nombre_del_paquete_descargado.tar.gz
cd tinker
cp linux/gfortran/* source/

mkdir bin
cd fftw/

export CC=gcc
export F77=gfortran

#si hemos intentado instalarlo antes ejecutamos el siguiente comando para limpiar esa instalacion
make distclean

./configure --prefix=<direcion donde esta tinker>/fftw --enable-threads
make
make install

cd ../source/

#comentar o eliminar las lineas 84 y 85 del fichero initial.f
call kmp_set_stacksize_s (2**28)
call kmp_set_blocktime (0)

#si hemos intentado instalarlo antes ejecutamos el siguiente comando para limpiar esa instalacion
rm -rf *.o *.a *.x

cd ../fftw/
cp lib/libfftw3_threads.a ../source/
cp lib/libfftw3.a ../source/

cd ../source/

./compile.make
./library.make
./link.make
./rename.make
```

Ilustración 18. Pasos a seguir para instalar Tinker. Imagen de elaboración propia.

Una vez seguidos estos pasos y siempre y cuando que no hayamos tenido ningún tipo de fallo, en la carpeta *bin* que hemos creado anteriormente, estarán todos los ejecutables de Tinker. No usamos todo Tinker, sino sólo algunos de sus ejecutables que luego sí usamos dentro de nuestro algoritmo, éstos son: *analyze*, *protein* y *xyzpdb*. Lo que hacemos es copiar estos ejecutables en la carpeta *bin* de nuestro algoritmo.

7.2.2. ProFit

ProFit está diseñado para ser un programa capaz de realizar ajustes por mínimos cuadrados de dos o más estructuras proteicas. Realiza una función simple y básica, pero permitiendo tanta flexibilidad como sea posible al realizar este procedimiento. Por ejemplo, se pueden especificar subconjuntos de átomos a considerar, especificar zonas que se ajustarán por número, secuencia o alineación de secuencia.

La instalación del ProFit es muy sencilla. En primer lugar, tenemos que descargarnos el código del siguiente enlace: *http://www.bioinf.org.uk/software/swreg.html*, nos tenemos que descargar la versión ProFit V3.1. Una vez que se descargue el comprimido, en nuestro caso sólo tuvimos que descomprimirlo y ejecutar *make* en la raíz de esa carpeta descomprimida. Con esto ya generará el ejecutable.

Ya obtenido el ejecutable, en nuestro caso lo que hemos realizado es un pequeño programa que vaya obteniendo cada uno de los resultados que nos ha generado nuestro algoritmo; Y por cada resultado obtenido lo comparamos con la proteína original (estructura nativa de la misma) mediante el programa ProFit. Con esta comparación obtenemos el RMSD, el cual guardamos en un fichero por cada solución.

8. *Multiobjective artificial bee colony* (MOABC)

Llegados a este punto, ya hemos explicado la optimización multiobjetivo (apartado 5) y el algoritmo ABC (sección 4.2), ahora en este apartado explicaremos la versión de MOABC y la optimización multiobjetivo que hemos decidido aplicar en este trabajo final de grado.

Empezaremos por la versión del MOABC que hemos aplicado (ver pseudocódigo en la página siguiente). El algoritmo comienza inicializando de forma aleatoria parte de la población, la parte correspondiente a las abejas obreras (si la población constase de 100 miembros la parte de las obreras es 50 y las abejas observadoras 50). Por cada individuo, los ángulos de torsión se generan aleatoriamente a partir de las regiones de restricción (tabla 2). Después se evalúa la energía de la conformación. En primer lugar, la estructura de la proteína en coordenadas internas (ángulos de torsión) se transforma en coordenadas cartesianas. Entonces, el potencial de energía CHARMM de la estructura se calcula utilizando rutinas de *Tinker Molecular Modeling Package*.

Tabla 2. Regiones correspondientes de las restricciones de las estructuras secundaria y supersecundaria.

	Restricciones de estructura supersecundaria	*Restricciones de estructura secundaria*
H (hélice alfa)	[-75º, -55º]	[-50º, -30º]
E (lámina plegada beta)	[-130º, -110º]	[110º, 130º]
a	[-150º, -30º]	[-100º, 50º]
b	[-230º, 30º]	[100º, 200º]
e	[30º, 130º]	[130º, 260º]
l	[30º, 150º]	[-60º, 90º]
t	[-160º, -50º]	[50º, 100º]
Indefinido	[-180º, 0º]	[-180º, 180º]

MOABC - pseudocódigo

```
AO ← generamos de forma aleatoria la primera mitad de la población. Abejas obreras.
num_evaluations ← 0, iter ← 0, flag ← 1
while(flag){
        #abejas obreras
        for j = 0 to tamañoPoblacion/2 do
                abejaMutadaM1 ← mutateGlobal(AO[j], M1)
                abejaMutadaM2 ← mutateLocal(AO[j], M2)
                if abejaMutadaM2 < abejaMutadaM1 then
                        abejaMutadaM1 ← abejaMutadaM2
                if abejaMutadaM1 < AO[j] then
                        AO[j] ← abejaMutadaM1; AO[j].limit ← 0
                else
                        AO[j].limit ← AO[j].limit + 1
                end if
        end for

        #calculamos la probabilidad de cómo de buenas son las abejas obreras
        calculateProbability()

        #abejas observadoras
        for j = tamañoPoblacion/2 to tamañoPoblacion do
                AO[j] ← selecionamosAbejaObrera() //mediante ruleta
                abejaMutadaM1 ← mutateGlobal(AO[j], M1)
                abejaMutadaM2 ← mutateLocal(AO[j], M2)
                if abejaMutadaM2 < abejaMutadaM1 then
                        abejaMutadaM1 ← abejaMutadaM2
                if abejaMutadaM1 <= AO[j] then
                        AO[j] ← abejaMutadaM1; AO[j].limit ← 0
                else
                        AO[j].limit ← AO[j].limit + 1
                end if
        end for

        #abejas exploradoras
        for j = 0 to tamañoPoblacion do
                if AO[j].limit > MaxLimit then
                        AO[j] ← Reinicializar la abeja
                end if
        end for

        #mejores abejas/soluciones
        AO ← fastNonDominatedSort(AO, tamañoPoblacion)
        AO ← crowdingDistanceAssignment(AO, tamañoPoblacion)
        AO ← selectBetterIndividuals(AO, (tamañoPoblacion/2))
        #guardar las soluciones que no son dominadas por nadie de la población
        archive_poblacion(); iter = iter + 1

        if (num_evaluations >= Tmax) || (iter > max_iterations) then
                flag ← 0
        end if
}
```

Ahora si proseguimos con la explicación de nuestro algoritmo después de una vez inicializada la población. A partir de este punto entramos ya en el bucle principal. Lo primero que se realiza es la mutación de las abejas obreras, para ello a cada abeja le aplicamos dos operadores de mutación distintos (M1 y M2, ecuación 16).

Ecuación 16. Fórmulas de los operadores de mutación utilizadas en este algoritmo. Estas fórmulas están sacadas de V. Cutello and others. A multi-objective evolutionary approach to the protein structure prediction problem. 2005.

$$M1 = \exp\left\{\frac{-2*numEvaluations}{Tmax}\right\} \qquad M2 = 1 + \frac{genes}{4} * M1$$

Donde Tmax es el número máximo de evaluaciones que vamos a realizar, numEvaluations es la evaluación que estamos realizado y genes es el número de residuos de la proteína que estamos tratando.

Pero estos operadores no hemos decidido utilizarlos de la misma manera. El primer operador M1, actúa a nivel global, es decir, puede cambiar drásticamente la conformación. Cuando este operador actúa sobre una cadena peptídica, todos los valores de los ángulos de torsión de la cadena principal se seleccionan de nuevo a partir de sus regiones restringidas correspondientes.

Por otro lado, el segundo operador M2, realiza una búsqueda local del espacio conformacional. Perturba algunos ángulos de torsión de un residuo elegido al azar. La probabilidad de mutación, para ambos operadores, disminuye a medida que vamos avanzando en el número de evaluaciones.

Cada vez que mutamos o inicializamos una solución debemos evaluar la nueva solución obtenida lo que significa incrementar el número de evaluaciones realizadas.

Una vez que hemos aplicado los dos operadores, vemos cuál de las dos soluciones domina a quien, en nuestro caso las abejas obreras hacen una comparación de

dominación estricta. El ganador de esta comparación se comparará ahora con la abeja obrera sin mutar, se realiza el mismo tipo de comparación. Finalmente sustituimos la abeja obrera por la ganadora de esta última comparación. Esto se realiza para cada una de las abejas obreras. Una vez que hemos terminado con todas las abejas obreras, tenemos que ver cómo es de buena cada abeja obrera respecto el resto de ellas, y darles más probabilidad de uso a las mejores abejas, esto lo realizamos para las siguientes abejas que entran en juego, las abejas observadoras.

Como ya hemos comentado en el párrafo anterior, las abejas que ahora entran en juego son las abejas observadoras. Cada una de ellas va a sacar un número aleatorio, por el cual mediante la elección por ruleta van a seleccionar una abeja obrera a la que copian, una vez hecho esto, realizan el mismo proceso de mutación que las abejas obreras con una distinción, el tipo de dominación que se realiza en las comparaciones es una dominación débil.

Una vez llegados a este punto, entran en acción las abejas exploradoras. La labor de las exploradoras es comprobar que ninguna de las abejas anteriores se haya quedado atascada/estancada en una solución (ésta es la utilidad del campo/contador *limit*, que permite chequear si el número de intentos sin mejorar una solución supera cierto límite, *MaxLimit*). Y sólo para aquellas abejas atascadas, lo que se les hace es "resetearlas", es decir, se les realiza el mismo proceso que al inicializar la población, pero esta vez, además, una vez inicializadas se les realizará una vez el proceso de mutación con dominancia estricta.

Tras esto, ahora toca la selección de las mejores abejas, es decir, tenemos que seleccionar las mejores entre todas las abejas y las que sean seleccionas serán las abejas obreras en el siguiente ciclo (iteración) del bucle. En nuestro caso, el criterio de selección que hemos decidido es seleccionar a las que sean dominadas por el menor número de abejas y dentro de las que tengan el mismo número de dominadores nos quedamos con las abejas que ofrezcan una mejor dispersión en el frente de Pareto (mayor distancia de *crowding*).

Finalmente se evalúa la condición de salida del bucle, en nuestro caso, puede ser por cumplir el número de evaluaciones máximo o llegar al máximo del número de iteraciones.

9. Paralelización de nuestra propuesta

Como ya se comentó anteriormente, el proceso de predicción de la estructura de las proteínas es un proceso costoso y tarda mucho tiempo. Por esta última característica nos vimos en la necesidad de tener que reducir el tiempo de cómputo. Por ello, se decidió introducir en este trabajo final de grado una parte de paralelismo.

Para abordar el paralelismo, nos hemos enfrentado a dos tareas, la paralelización con OpenMP y la eliminación de los conflictos de E/S existentes en el programa.

9.1. Paralelización con OpenMP

Aquí se abordará la implementación de OMP que se ha realizado. Lo primero que se realizó es un estudio de en qué zonas del algoritmo se produce más desgaste temporal.

Para realizar este estudio se dividió el algoritmo (su bucle) en: abejas obreras, cálculo de la probabilidad, abejas observadoras, abejas exploradoras y selección de las mejores abejas. Se pudo observar que el 93,478% del tiempo total se empleaba en las zonas de abejas obreras y observadoras, y las abejas exploradoras suponían un 6,507% del tiempo, es decir, las abejas ocupaban el 99,985% del tiempo total.

Tras realizarse este estudio más en profundidad, nos dimos cuenta que todo el tiempo que consumían las abejas estaba en la función en la que llamamos a los módulos de Tinker.

Una vez que se ha encontrado donde está el mayor consumo de tiempo (perfilado del tiempo de ejecución del programa), ya podemos aplicar OMP. En concreto, las sentencias de OMP se han aplicado en los bucles for que manejan a las abejas. La estructura general de las directivas que se han aplicado es la siguiente:

#pragma omp parallel for firstprivate (lista de variables) schedule(dynamic)

Cada vez que aparece esta directiva lo que hace es abrir una región paralela de hasta x hilos de ejecución (según los que se hayan establecido, en nuestro caso son 64 hilos), y cada hilo tendrá una copia privada de cada una de las variables; y si alguno de los hilos terminase con sus tareas asignadas y aun hubiese tareas por ejecutar se le asignarán nuevas tareas (planificación dinámica).

Se podría decir que aquí concluye la parte de paralelismo, pero nos encontramos con unos conflictos a la hora de ejecutar de forma paralela la parte donde se llamaban a los módulos de Tinker. Esto se explicará en la siguiente sección.

9.2. Soluciones de conflictos de E/S

En esta sección se explicarán los problemas que tuvimos con la E/S al paralelizar y la forma en la que los solucionamos.

Los problemas surgen por la forma de funcionar de los ejecutables que se usan de Tinker. Y es que ellos interactúan con la pantalla y el teclado. Para solucionar este problema lo que se realizó es redirigir ese flujo de E/S.

Respecto al flujo de salida, la pantalla, en vez de que se muestren los datos, hicimos que se generasen ficheros con nombres característicos del ejecutable que los generase. Y respecto al flujo de entrada, el teclado, se cambió la forma de introducir los datos de tal forma que proporcionando un fichero ya supiese leer los datos del mismo.

De esta forma, se consiguió solucionar los problemas de E/S, dado que en la ejecución en paralelo no se disponen de varias pantallas y teclados. Con esta solución y añadiendo un identificador por cada hilo a los ficheros que se necesitan/generan

para el tratado de los ejecutables de Tinker, con esto ya tenemos el algoritmo paralelizado y funcional.

10. Ajustes del algoritmo

En este apartado hablaremos de la batería de pruebas que hemos realizado (a los parámetros configurables) para ajustar de la manera más óptima el algoritmo.

La proteína con la que hemos ejecutado todas estas pruebas es la 1ZDD y para cada prueba hemos realizado 10 repeticiones, donde luego hemos sacado la media de energía y tiempo de ejecución. Además, el número de evaluaciones (de las funciones objetivo) que les hemos exigido a las pruebas es 50.000, un valor lo suficientemente alto como para obtener resultados fiables y al mismo tiempo tener un gasto computacional moderado-alto, pero no tan alto como si de una prueba real (con 500.000 evaluaciones, diez veces más) se tratase.

En este apartado se utiliza la descripción del algoritmo detallada en el apartado 8 (MOABC). Recordemos que la mejor energía es la menor. En cada prueba únicamente se modificará un campo configurable para ver qué efecto produce en los resultados, el resto de campos a excepción del que se esté estudiando en ese momento se mantienen igual. El tamaño de la población para todas las pruebas es de 8.

10.1. Tipo de dominación de las abejas exploradoras

Las abejas exploradoras una vez que generan una nueva solución al azar, para que esta nueva abeja pueda ser competitiva se muta. En este apartado vamos a plantear las dos posibles configuraciones para la dominación cuando se muta (para comparar si la abeja mutada es mejor que la original): dominación estricta o dominación débil. A continuación, se mostrará la tabla 3 con los resultados de las pruebas.

Tabla 3. Resultados de las pruebas del tipo de dominación de las abejas exploradoras.

Nº de la prueba	*Dominación débil*	*Dominación estricta*
1	-747,4053	-564,3944
2	-877,3309	-873,2914
3	-924,314	-961,5453
4	-500,413	-810,421
5	-913,4754	-460,8454
6	-823,9831	-251,8422
7	-878,7344	-1045,8366
8	-1036,8387	-1016,1531
9	-937,0603	-1017,9948
10	-347,2987	-1101,9108
Media de la energía	-798,68538	-810,4235
Media de tiempo(s)	56211,73706	51460,86123

Como podemos observar en la tabla 3, los mejores resultados nos lo ofrece la configuración de dominación estricta. Que no sólo nos aporta mejores energías, sino que también mejora el tiempo de ejecución, haciendo que tarde una hora y veinte minutos menos aproximadamente que la dominación débil. Por tanto, nos quedamos con la configuración que ofrece mejores resultados.

10.2. Fórmulas de MOfitness

Como ya hemos mencionado, el fitness determina la calidad de una solución, es decir, lo buena que es una solución. El MOfitness es lo mismo, pero para los algoritmos multiobjetivo.

En esta sección hablaremos sobre las distintas fórmulas que hemos probado para calcular la calidad de una solución y cuáles son los resultados de aplicar cada fórmula.

A continuación, mostraremos todas las fórmulas (ecuaciones 17 a 22) que hemos usado para esta prueba.

Ecuación 17. Ecuación denominada original. Esta ecuación está sacada de Álvaro Rubio Largo. Multiobjective Metaheuristics and Parallel Computing for Optimizing WDM Optical Networks. Tesis Doctoral. Universidad de Extremadura. 2013.

$$Original = \frac{1}{2^{rank} + \frac{1}{1 + distancia}}$$

Ecuación 18.

$$Versión\ 2 = \frac{rankMax - rank}{rankMax} + enerNormalizada$$

Ecuación 19.

$$Versión\ 3 = \frac{rankMax - rank}{rankMax} + \frac{distancia}{distanciaMax}$$

Ecuación 20.

$$Versión\ 4 = \frac{rankMax - rank}{rankMax} + \frac{distancia}{distanciaMax} + enerNormalizada$$

Ecuación 21. Esta ecuación, junto con las tres ecuaciones anteriores, son de elaboraciones propias.

$$Versión\ 6 = \frac{1}{2^{rank}} + enerNormalizada$$

Ecuación 22. Esta ecuación es de elaboración propia pero combinada con la ecuación denominada original.

$$Versión\ 5 = Original + enerNormalizada$$

Donde para cada solución, rank es el ranking del frente de Pareto donde se encuentra esa solución (un rank menor indica un mejor frente de Pareto), distancia es el resultado de aplicar crowdingDistance, enerNormalizada es la energía de esta solución, pero normalizada, rankMax y distanciaMax son el rank y distancia máxima de la población.

Cada una de estas fórmulas plantea diversas formas de valorar la calidad de una solución, pero si nos fijamos las combinaciones son entre el rank (calidad basada en dominancia), distancia (calidad basada en crowding, dispersión) y la energía (calidad basada en una métrica biológica). Recordemos que mientras mayor sea el resultado de cualquiera de las formulas mejor es dicha solución.

En la siguiente tabla 4 veremos los resultados obtenidos por todas las fórmulas que hemos visto anteriormente.

Tabla 4. Resultados de las pruebas para cada una de las diferentes fórmulas del MOfitness.

N.º de prueba	*Original*	*Versión 2*	*Versión 3*	*Versión 4*	*Versión 5*	*Versión 6*
1	-564,3944	-1027,5466	-1045,7672	-740,2343	96,185	-892,3395
2	-873,2914	-912,5981	-610,6781	-573,7091	-1126,3784	-766,0125
3	-961,5453	-1064,4005	-865,0971	3107,0159	-806,7364	-375,4436
4	-810,421	-632,5314	-670,8523	-369,6741	-900,8235	-509,2562
5	-460,8454	-791,0135	-822,8409	-896,7781	-705,3514	-319,901
6	-251,8422	-698,6936	-734,9907	-683,9215	-704,0579	-717,5343
7	-1045,8366	-1057,4595	-934,8411	-706,7333	-942,1778	-519,8638
8	-1016,1531	-772,5261	-565,9815	-236,3474	-964,4007	-221,1128
9	-1017,9948	-876,8333	-77,7358	-892,988	-871,4931	-903,0606
10	-1101,9108	1296,7755	-26,5198	-772,3374	-796,8427	-658,785
Media de energía	-810,4235	-653,68271	-635,53045	-276,57073	-772,20769	-588,33093
Media de tiempo(s)	51460,86123	56252,69729	51993,5267629	56168,99205	52024,752669	56282,15921

Las dos únicas fórmulas que lo combinan todo son a su vez la peor y la solución más cercana a la mejor (versión 4 y 5) respectivamente. Cabría pensar que estas dos fórmulas serían las que mejores resultados nos darían al estar más completas o al menos que sus resultados serían parecidos, pero como vemos en los resultados esto no es así. Por otro lado, el resto de las formulas sólo utilizan dos de los tres posibles componentes.

La versión 2 y 3 tienen una parte común (rank), como vemos los resultados son muy similares, pero la versión 2 que implementa la energía en vez de la distancia (versión 3), nos ofrece resultados un poco mejores a pesar de un tiempo peor. La versión 6 es una forma diferente, pero con los mismos factores de combinación que la versión 2, nos da resultados bastantes peores que dicha versión y tiempos muy similares.

Finalmente, la versión que mejor media energética y de tiempo nos ofrece es la original. La cual combina el rank y la distancia crowding para determinar la calidad de las soluciones. Podemos observar que la versión 3 hace la misma combinación, pero de forma distinta y los resultados no se parecen en nada, lo más parecido son los resultados de tiempo. Y la versión 5 utiliza la misma fórmula además de la energía normalizada lo que parece que empeora los resultados.

En definitiva y tal y como se puede observar en la tabla, la configuración de la fórmula con la que nos quedamos es la original, debido a que es la que mejores resultados nos ofrece, tanto en energía como en tiempo.

10.3. Tipo de ordenación

En este apartado hablaremos sobre qué tipo de ordenación nos da mejores resultados, pero primero debemos comentar dónde se aplica el método de ordenación. Dentro del método de selección, se seleccionan las abejas que sean dominadas por el menor número de abejas y dentro de las que tengan la misma dominancia (las del mismo frente de Pareto), tenemos que elegir algún método para desempatar entre ellas (ordenarlas de alguna forma).

Una primera alternativa es, dentro de un frente de Pareto, quedarnos con aquellas abejas que nos ofrecen una buena dispersión del frente. En particular, nosotros usamos la *crowdingDistance* (Deb et al., 2002). Básicamente lo que hace este método es calcular, para cada punto del frente en cuestión, la distancia media de dos puntos, uno a cada lado del punto en cuestión. El resultado es una estimación del perímetro por los vecinos más cercanos al punto. Una vez calculadas todas las distancias del frente aquí es donde tenemos que ordenar el frente, quedándonos con aquellas con tienen mejor (mayor) distancia de crowding. Esta es la ordenación original propuesta en (Deb et al., 2002).

Una segunda alternativa es, dentro de un frente de Pareto, quedarnos con aquellas abejas que nos ofrecen la mejor energía. Es decir, en este segundo caso las ordenaríamos por su valor de energía. Véanse los resultados obtenidos para ambas alternativas en la siguiente tabla 5.

Tabla 5. Resultados de la prueba de los distintos tipos de ordenación.

N.º de prueba	*Ordenación original*	*Ordenación por energía*
1	-564,3944	-958,7765
2	-873,2914	-1082,9157
3	-961,5453	-796,1115
4	-810,421	-656,7591
5	-460,8454	-865,963
6	-251,8422	-916,0044
7	-1045,8366	-969,7687
8	-1016,1531	-866,6097
9	-1017,9948	-895,4068
10	-1101,9108	-869,4496
Media de la energía	-810,4235	-887,7765
Media de tiempo(s)	51469,86123	32531,70545

Esta es la primera prueba que además de ganar en energía ganamos en tiempo de ejecución, es decir, que tarda menos tiempo. Energéticamente la ordenación por energía nos aporta una pequeña mejora, de unos 77 puntos, pero además la mejora en tiempo sí que es bastante notable, ya que pasa de las 14 horas a las 9 horas gracias a la ordenación por energía. Por ello, y de forma muy clara, decidimos elegir la ordenación por energía.

10.4. Nº de mutaciones de las abejas exploradoras

Anteriormente hemos realizado pruebas para ver qué tipo de dominancia aplicar a las abejas exploradoras. Ahora nos toca decidir cuantas veces se mutan estas abejas. Este valor les puede otorgar de más o menos competitividad respecto al resto de la población. Mientras más se mute, más competitiva puede ser una solución. Esta prueba la hemos realizado con los siguientes valores (cantidad de mutaciones): 1, 5, 10, 15, 20, y los resultados los podemos observar en la tabla 6.

Tabla 6. Resultados de la prueba del nº de veces que se muta una nueva abeja para otorgarle competitividad.

N.º de prueba	*1*	*5*	*10*	*15*	*20*
1	-656,9863	-958,7765	-563,8522	-762,1577	-1054,3215
2	-1009,3522	-1082,9157	-949,9664	-512,9627	-794,0221
3	-737,5981	-796,1115	-979,8873	-969,915	-1037,3978
4	-993,1307	-656,7591	-858,0457	-1039,8947	-1060,8552
5	-1013,201	-865,963	-804,6979	-977,3375	-873,3712
6	-993,3107	-916,0044	-530,1414	-954,8104	-937,6657
7	-724,5015	-969,7687	-752,6336	-1159,5976	-952,8496
8	-1077,3876	-866,6097	-890,961	-687,3671	-770,7224
9	-1027,8219	-895,4068	-866,5633	-807,7748	-453,968
10	-988,0783	-869,4496	-926,6064	-1004,2803	-807,0207
Media de la energía	-922,13683	-887,7765	-812,33552	-887,60978	-874,21942
Media de tiempo(s)	32530,92067	32531,70545	32584,999	31874,54025	31948,83283

Aquí vemos que el mejor resultado lo da el valor 1 (sólo una mutación) y los demás son más o menos iguales, el valor 10 es el peor. Pero esto de que la prueba con valor 1 sea la mejor tiene un porqué, y es que si gastamos evaluaciones para que las nuevas abejas intenten ser competitivas, esas evaluaciones gastadas no podrán ser utilizadas para que las abejas que ya son competitivas lo sean aún más. En conclusión, la mejor opción es utilizar el valor 1 (sólo una mutación para cada nueva abeja).

10.5. Nº de evaluaciones para que sea estancamiento

Siguiendo con las abejas exploradoras, lo siguiente a considerar es con cuantas evaluaciones (intentos de mejora) consideramos que una abeja debe de ser "reseteada", es decir, dónde está el límite. Este límite determina cuándo consideramos que una abeja se ha quedado estancada. Las pruebas se han realizado con estos valores: 10, 15, 20, y los resultados pueden observarse en la tabla 7.

Tabla 7. Resultados de la prueba del nº del límite para considerarlo estancamiento.

N.º de prueba	*10*	*15*	*20*
1	-1004,433	-656,9863	-957,4381
2	-907,4663	-1009,3522	-881,8706
3	-739,3876	-737,5981	-960,1923
4	-982,121	-993,1307	-1019,4594
5	-936,2203	-1013,201	-933,3578
6	-876,6232	-993,3107	-1180,0299
7	-942,5754	-724,5015	-349,0264
8	-832,5009	-1077,3876	-545,0414
9	-711,3096	-1027,8219	-971,8068
10	-715,2138	-988,0783	-959,9023
Media de la energía	-864,78511	-922,13683	-875,8125
Media de tiempo(s)	32565,30379	32530,92067	31805,82052

Podemos observar que el tiempo es parejo en todas las configuraciones y que las configuraciones de 10 y 20 dan resultados de energía similares, siendo un poco mejor la configuración de 20. Sin embargo, el mejor resultado energético nos lo ofrece una configuración intermedia entre las dos anteriores, la configuración de 15 intentos como límite para detectar el estancamiento. Esta es la configuración con la que nos quedamos.

10.6. Tipo de dominación de las abejas observadoras

Aquí realizaremos el mismo tipo de prueba del tipo de dominación de las abejas exploradoras pero esta vez sobre las abejas observadoras. La tabla 8 muestra los resultados de los experimentos para esta prueba.

Tabla 8. Resultados de la prueba de tipo de dominación en las abejas observadoras.

N.º de prueba	*Dominación débil*	*Dominación estricta*
1	-656,9863	-712,6156
2	-1009,3522	-1056,8997
3	-737,5981	-885,0968
4	-993,1307	-794,1906
5	-1013,201	-936,2571
6	-993,3107	-24,3015
7	-724,5015	-717,2309
8	-1077,3876	-693,5954
9	-1027,8219	-881,3496
10	-988,0783	-702,7478
Media de la energía	-922,13683	-740,4285
Media de tiempo(s)	32531,70545	40387,81288

Podemos observar como aplicando una dominación débil en la comparación tras la mutación de las abejas observadoras, se obtienen resultados bastantes más favorables que aplicando la dominación estricta. Esto puede ser debido a que siendo más estrictos se cojan menos veces las abejas mutadas, teniendo más soluciones iguales

(repetidas) y provocando un aumento en el contador de estancamiento de cada abeja, lo que también conlleva que se reseteen antes, produciendo todo esto un avance más lento del algoritmo.

En definitiva, dado que con la configuración de dominación débil obtenemos no sólo mejores energías sino además una reducción del tiempo de ejecución, ésta es la configuración que decidimos aplicar.

10.7. Operadores de mutación (M1 y M2)

Las fórmulas que se han descrito en el apartado de MOABC (apartado 8) para M1 y M2 (Cutello et al., 2006) las denominaremos como la fórmula original. En este apartado mostraremos los resultados con distintas fórmulas de M1 y M2, las cuales en su momento nos hicieron pensar que podrían darnos buenos resultados. La fórmula de M2 permanecerá constante ya que ella siempre depende de M1 (ecuación 23), por lo tanto, las fórmulas que veremos serán sólo las de M1. Las fórmulas con las distintas variantes de M1 (ecuaciones 24-26) son las siguientes.

Ecuación 23. Ecuación del segundo operador de mutación. Esta ecuación está sacada de V. Cutello and others. A multi-objective evolutionary approach to the protein structure prediction problem. 2005.

$$M2 = 1 + \frac{genes}{4} * M1$$

Ecuación 24.

$$versión1 = M1 = \exp\left\{\frac{-2 * iter}{iterMax}\right\}$$

Ecuación 25.

$$versión2 = M1 = \exp\left\{-1 * \left(\frac{enerMax - ener}{enerMax - enerMin}\right)\right\}$$

Ecuación 26. Esta ecuación, junto con las dos anteriores, son de elaboraciones propias.

$$versión3 = M1 = \exp\left\{-1 * \left(\frac{ener - enerMin}{enerMax - enerMin}\right)\right\}$$

Donde iter es la iteración actual por la que va el algoritmo e iterMax es el número de iteraciones máximo que se pueden hacer. Ener es la energía de la abeja en cuestión y enerMax la energía máxima de la población y enerMin la energía mínima de la población.

Cada una de las versiones no sólo implica cálculos aritméticos distintos, sino que dada la utilidad que tienen los operadores de mutación, tiene efectos distintos. La versión 1 a diferencia de la original, lo que produce es más mutaciones por ciclo, igual que la original disminuye a medida que vamos avanzando en el número de iteraciones, pero este número de iteraciones avanza más bruscamente que el de evaluaciones.

Por su parte, las versiones 2 y 3 no afectan al número de mutaciones general sino a qué abejas se mutan más. La versión 2 lo que produce es que las peores abejas muten más que las demás, dándoles así más oportunidades de ser competitivas. Por el contrario, la versión 3 produce que las mejores abejas muten más, es decir, damos más probabilidades de que sean mejores las que ya de por sí son las mejores de la población, provocando que las peores sean las que menos muten y acaben siendo descartadas.

Ahora, en la siguiente tabla 9, veremos los resultados obtenidos para cada una de las fórmulas de los operadores de mutación.

Tabla 9. Resultados de la prueba de las distintas ecuaciones de los operadores de mutación.

N.º de prueba	*Original*	*Versión 1*	*Versión 2*	*Versión 3*
1	-656,9863	-779,4387	-866,4114	-953,9345
2	-1009,3522	-777,1306	-783,6461	-827,0211
3	-737,5981	-964,3371	-933,2791	-883,7918
4	-993,1307	-429,5083	-909,8485	-975,368
5	-1013,201	-762,0966	-945,3339	-837,8744
6	-993,3107	-600,6363	-828,1242	-941,0611
7	-724,5015	-927,0586	-1063,2005	-820,0066
8	-1077,3876	-791,0042	-1009,2504	-870,914
9	-1027,8219	-976,3459	-1018,4475	-902,8974
10	-988,0783	-836,2746	-733,1928	-941,221
Media de la energía	-922,13683	-784,38309	-909,07344	-895,40899
Media de tiempo(s)	32531,70545	29963,61912	30974,72704	32220,94933

Observando los datos de la tabla, la versión original es la que mejores resultados energéticos nos muestra. Dado que un 40% de los resultados supera el -1000 y sólo un 30% es menor de -900 la media que nos ofrece esta versión es muy favorable. En conclusión, nos quedamos con la versión original de estas fórmulas.

También hay que decir que, respecto a los tiempos de ejecución, esta versión no es la mejor del grupo, sino que, al contrario, siendo nuestra versión elegida unos 26 minutos aproximadamente más lenta respecto a la siguiente versión con mejores resultados energéticos.

Aquí concluyen todos los resultados de las pruebas de ajuste del algoritmo. Todavía queda por configurar otro parámetro, el tamaño de la población. Pero este parámetro está dentro del siguiente apartado, los resultados del paralelismo.

10.8. Coste computacional de todos los ajustes realizados

Antes de acabar con este apartado, cabe mencionar el coste temporal total que habría tenido este apartado. Para ello utilizaremos los tiempos medios y los tenemos en las tablas 10 y 11.

Tabla 10. Tiempo total de cada una de las pruebas.

Tiempo total (s)	Prueba
1256910,0094	Operadores de mutación (M1 y M2)
729195,1833	Tipo de dominación de las abejas observadoras
969020,4498	Nº de evaluaciones para que sea estancamiento
1614709,982	Nº de mutaciones de las abejas exploradoras
840015,6668	Tipo de ordenación
3241829,892119	Formulas del MOfitness
1076725,9829	Tipo de dominación de las abejas exploradoras

Tabla 11. Tiempo total de todas las pruebas.

Tiempo total del apartado de ajustes del algoritmo		
9728407,1663 (s)	2702,3353 (h)	112,5973 (días)

Como puede apreciarse, el tiempo es más que considerable. A este tiempo habría que sumarle el del apartado siguiente, para así de esta forma tener el tiempo completo que se habría invertido en el ajuste del algoritmo.

11. Resultados de la paralelización

En este apartado vamos a hablar más que de resultados en sí, de las pruebas que hemos realizado para ajustar el tamaño de la población jugando con el grado de paralelismo. En definitiva, veremos qué resultados obtenemos aplicando distintos grados de paralelismo. No sólo nos fijaremos en la mejora de tiempo, sino también en cómo afecta a la energía (a los resultados).

Como ya hemos comentado anteriormente, el parámetro configurable que afecta al tiempo de cómputo es el tamaño de la población; porque como ya hemos mencionado antes, el mayor gasto de tiempo está en las fases que gestionan a las abejas, es decir, a la población. A diferencia de las pruebas de ajuste del algoritmo, en este caso el número de evaluaciones sí son ya las definitivas, y para el caso de la proteína 1ZDD dicho número es de 500.000 evaluaciones de las funciones objetivo.

En este caso lo que nos importa es reducir el tiempo de cómputo, pero siempre sin empeorar la energía (los resultados), recordemos que lo que más nos interesa es tener la mejor energía posible.

Los tamaños de población con los que hemos hecho pruebas son: 2, 4, 8, 16, 32, 64 y 128. El resto de parámetros configurables es el mismo para cada una de las pruebas, y fueron explicados en el apartado previo. Tras la realización de las pruebas vemos que los valores de energía no son afectados de forma negativa. En la siguiente tabla (tabla 12) se muestra para cada experimento: el número de la prueba (NP), la media de energía (ER), la media de tiempo (T), la eficiencia (E) y el *speedup* (S). Recordemos que la prueba de población=2 es nuestra versión secuencial (puesto que un individuo de la población será una abeja obrera y el otro una abeja observadora) y que nuestro hardware admite hasta 64 hilos de ejecución (al disponer de 64 cores), por ese motivo, el máximo teórico en el tamaño de la población es 128 (que al dividirlo entre 2 nos generará 64 abejas obreras y 64 abejas observadoras).

Tabla 12. Resultados del experimento de tamaño de la población junto con el tiempo medio, speedup y la eficiencia según el nº de hilos utilizados en cada prueba. El nº de hilos es la mitad de la población de esa prueba.

NP	*2*	*4*	*8*	*16*	*32*	*64*	*128*
1	-960,5469	-852,6517	-1108,1387	-883,701	-1050,5993	-1094,0777	-1154,7092
2	-981,4805	-958,5856	-992,963	-1008,6433	-1105,1969	-1071,3662	-1042,0555
3	-855,2649	-1017,6709	-1057,5225	-885,5414	-1006,9623	-989,0148	-1087,083
4	-958,5278	-748,372	-996,2705	-804,441	-980,5177	-1100,7162	-1059,6361
5	-964,9712	-948,5379	-960,5814	-1053,3743	-1174,0745	-1126,5002	-1117,0525
6	-1034,6167	-1089,4317	-987,964	-817,8346	-1068,4674	-898,5871	-1064,7738
7	-882,6996	-907,1174	-1024,7603	-543,13	-1084,6189	-1187,2323	-930,2917
8	-984,0056	-799,504	-990,4144	-898,9241	-810,2535	-1086,0316	-874,9859
9	-910,2953	-978,0938	-1057,0348	-932,3797	-1078,0984	-1054,4305	-1311,5298
10	-842,0521	-929,1039	-948,3451	-931,2065	-908,8521	-1069,1319	-1116,0159
ER	-937,44606	-922,90689	-1012,39947	-875,91759	-1026,7641	-1067,70885	-1075,81334
T(S)	862358,5566	689914,5279	436037,3918	217226,7149	108371,7347	59445,44302	33644,08239
S	1	1,25	1,98	3,97	7,96	14,51	25,63
E	1	0,625	0,495	0,49625	0,4975	0,45344	0,4005

Como podemos observar, los valores de eficiencia y *speedup* no son buenos del todo (aunque la eficiencia no suele estar muy por debajo del 50%). Esto es debido a la enorme gestión de ficheros realizada por el programa Tinker para el cálculo de las energías, y difícilmente mejorable mediante paralelismo (es decir, la parte secuencial del programa que no puede paralelizarse tiene un tiempo de ejecución grande, y según sabemos por la Ley de Amdahl esto va a limitar en gran medida la aceleración máxima posible). Recordemos que usamos este programa (Tinker, escrito en Fortran) para asegurar que el cálculo de las energías es totalmente correcto desde un punto de vista biológico. En todo caso, el hecho de aplicar mejoras de tiempo no es sólo por estos resultados, sino por el tiempo que supone hacer una prueba con población 2 ~ que es de 10 días, frente al tiempo que nos plantea la población de 128 ~ 9 horas. Por ello, a pesar de obtener eficiencias en torno al 50% vemos valiosos los resultados, manteniendo y aplicando paralelismo.

También cabe mencionar que el tiempo de cómputo (TC) aproximado que nos ha llevado obtener todos los resultados de este apartado sigue la siguiente fórmula:

$$TC = T_2 * 10 + T_4 * 10 + T_8 * 10 + T_{16} * 10 + T_{32} * 10 + T_{64} * 10 + T_{128} * 10$$

Esto quiere decir que TC= 24069984,51 segundos = 6686,10681 horas = 278,59 días. Éste es el tiempo si se hubiesen lanzado de una en una las pruebas. Pero lo que nosotros hemos hecho es lanzar tantas pruebas como pudiéramos sin superar el número de cores del nodo del clúster que hemos usado.

A este tiempo habría que sumarle el tiempo del apartado anterior, esto haría un total de 391,1873 días (más de un año).

12. Resultados biológicos y multiobjetivo

Antes de comenzar con este apartado, debemos mencionar la diferencia entre resultados biológicos y multiobjetivo. Cuando hablemos de energía, eso es un resultado multiobjetivo (calculado internamente por nuestro algoritmo multiobjetivo para saber la calidad de sus soluciones), mientras que cuando hablemos de calidad de la estructura final de la proteína (RMSD) eso es un resultado biológico.

El RMSD se ha calculado mediante el algoritmo de *McLachlan* (McLachlan 1982). Para poder utilizar este algoritmo, necesitamos tener la estructura nativa de cada proteína y las que vamos generando/obteniendo con nuestro algoritmo. Estas estructuras son ficheros con la extensión .pdb (es decir, están en el formato del *Protein Data Bank, PDB*). Nosotros las estructuras nativas las obtenemos mediante Tinker, realizando los siguientes pasos: nos descargamos la estructura de la proteína del PDB, la convertimos a coordenadas cartesianas (fichero con extensión .xyz) y después lo volvemos a pasar a formato .pdb.

El motivo de hacer esto es que hay diferencias de cabeceras (dentro de los ficheros) entre los que generamos con nuestro algoritmo y los descargados directamente del PDB. Por ello, utilizamos Tinker para que tengan la misma estructura interna los archivos y poder así compararlos correctamente.

A partir de aquí, se presentarán los resultados obtenidos utilizando ya pruebas finales (con nuestro algoritmo totalmente configurado). Aplicaremos nuestro algoritmo a un péptido corto (Met-encefalina o metionina) utilizado en numerosos artículos y luego a tres secuencias más de proteínas del Banco de Datos de Proteínas (PDB). Para las proteínas 1ZDD, 1UTG y 1CRN estableceremos el número máximo de evaluaciones a 5×10^5, mientras que para la Met-encefalina (MET) el número máximo de evaluaciones es $3{,}5 \times 10^5$.

Hay parámetros que varían en función de la prueba que estemos realizando, estos los explicaremos en detalle en cada una de las pruebas, el resto de parámetros

configurables son iguales para todas las pruebas: población = 128, por tanto, número de hilos en las zonas paralelas igual a 64.

Antes de empezar ya con los resultados, debemos mencionar que cuando nos refiramos a la mejor energía, no será la mejor de las 11 repeticiones de cada experimento sino la mediana una vez obtenida la mejor energía de cada repetición.

MET es un polipéptido muy corto con sólo 5 aminoácidos, es decir, es un pentapéptido. Se trata de una encefalina que interviene en la regulación del dolor y en la nocicepción corporal. Las encefalinas son endorfinas unidas al receptor opioide corporal. Fueron descubiertas en 1975, en dos formas, cada una de ellas compuesta por cinco aminoácidos, cuatro de ellos idénticos en ambos compuestos y una conteniendo leucina ("Leu") y la otra metionina ("Met", que es la que usamos aquí). Ambas son productores del gen de la proencefalina. Se cree que estos dos neuropéptidos pueden deprimir las neuronas del sistema nervioso central. Volviendo al péptido MET, se estima que tiene más de 10^{11} conformaciones localmente óptimas. Este péptido es un "banco de pruebas" evidente, para el que se han hecho una gran cantidad de experimentos en la literatura. La configuración de restricciones que hemos utilizado es la estructura secundaria y la secuencia de este polipéptido la hemos sacado del PDB, concretamente la anotación de STRIDE (asignación de estructura secundaria de proteínas basada en el conocimiento). En este caso cuando hablemos de la conformación, hemos utilizado 1PLW para el cálculo del RMSD ya que tiene la misma estructura que el MET. Para el MET, el valor energético más bajo del frente de Pareto mediana es 1,8813$kcal\,mol^{-1}$, el cual nos da un $RMSD_{C_\alpha} =$ 1,494; sin embargo, el mejor $RMSD_{C_\alpha}$ es 1,392, el cual nos lo da una energía de 5,1674 $kcal\,mol^{-1}$, estos datos se pueden observar de forma más clara en la tabla 13.

Tabla 13. La mejor energía y el mejor RMSD de MET.

	Energía $(kcal\ mol^{-1})$	$RMSD_{C_\alpha}$(Å)
Min energía	1,8813	1,494
Min RMSD	5,1674	1,392

En la ilustración 19 se puede observar la similitud entre la estructura 3D real de la proteína MET (imagen de la derecha) y la predicha gracias a nuestro algoritmo MOABC (imagen de la izquierda).

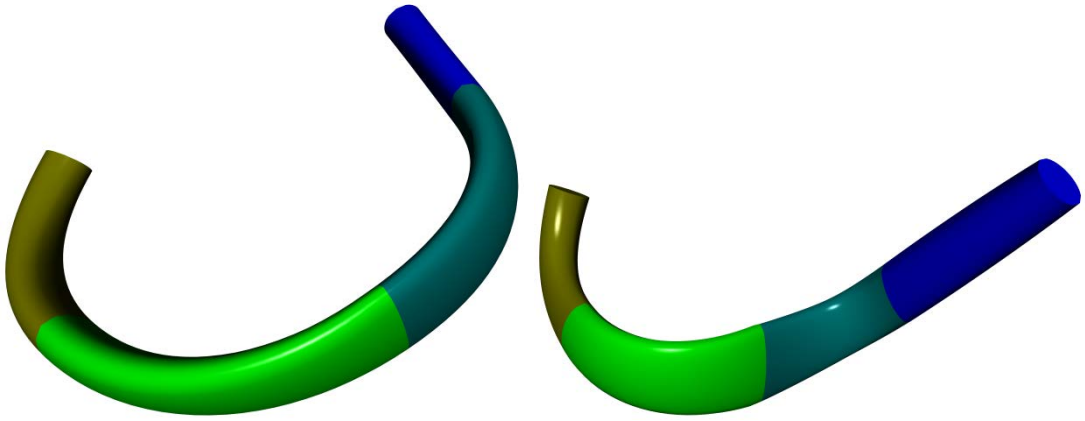

Ilustración 19. Conformaciones (estructura 3D) predicha (a la izquierda) y nativa (a la derecha) para la proteína MET ($RMSD_{C_\alpha}$ = 1,392 Å). Imágenes de elaboración propia gracias al programa BALLView.

1ZDD es un péptido de doble hélice de 34 residuos (aminoácidos). Para esta proteína, la información de la estructura secundaria nativa se determinó utilizando el servidor PDB original. Para el 1ZDD, el valor energético más bajo del frente de Pareto mediana es -1122,7198 $kcal\ mol^{-1}$, el cual nos da un $RMSD_{C_\alpha}$ =2,3. Sin embargo, el mejor $RMSD_{C_\alpha}$ es 2,211, el cual nos da una energía de -1063,4068 $kcal\ mol^{-1}$. Véase los resultados en la tabla 14.

Tabla 14. La mejor energía y el mejor RMSD de 1ZDD.

	Energía ($kcal\ mol^{-1}$)	$RMSD_{C_\alpha}$(Å)
Min energía	-1122,7198	2,3
Min RMSD	-1063,4068	2,211

En la ilustración 20 se puede observar la similitud entre la estructura 3D real de la proteína 1ZDD (imagen de la derecha) y la predicha gracias a nuestro algoritmo MOABC (imagen de la izquierda).

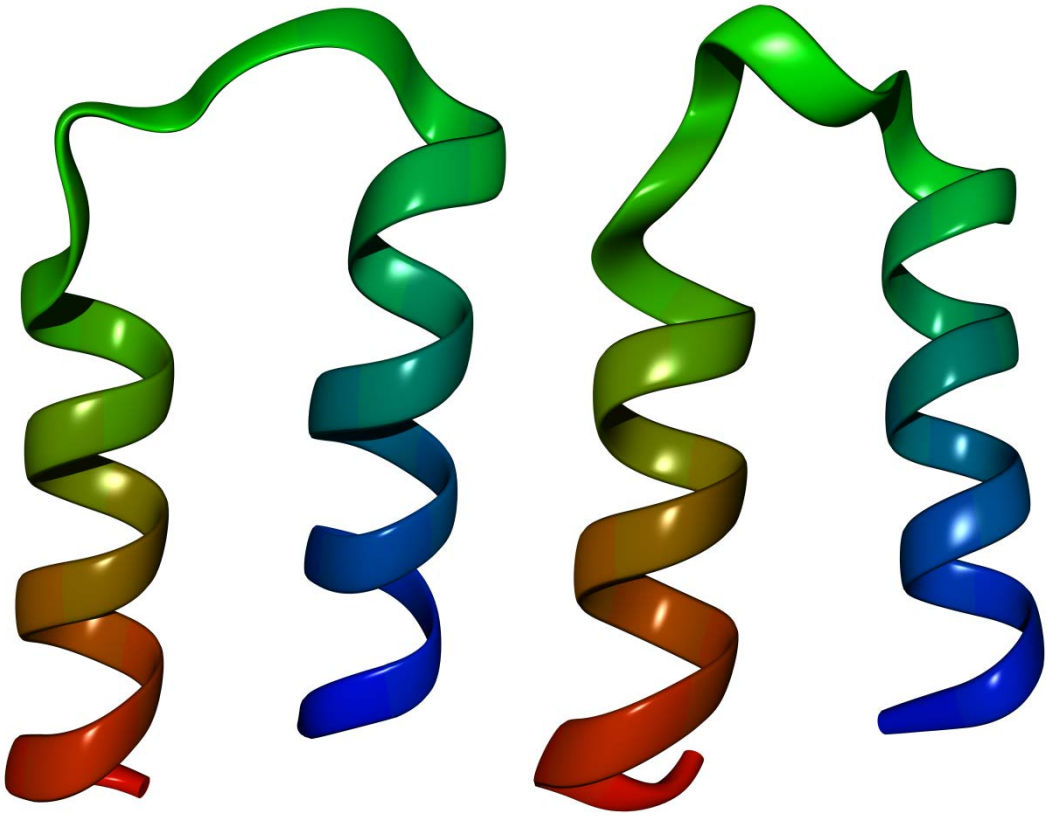

Ilustración 20. Conformaciones (estructura 3D) predicha (a la izquierda) y nativa (a la derecha) para la proteína 1ZDD ($RMSD_{C_\alpha}$ = 2,211 Å). Imágenes de elaboración propia gracias al programa BALLView.

Para los resultados de las pruebas de MET y de 1ZDD, podemos ver que todos presentan buenas características tanto en términos de RMSD como de energía. Para estas proteínas, el algoritmo es capaz de producir un conjunto de estructuras de proteínas de buena calidad. Esta buena correlación puede sugerirnos que minimizar

la energía variando la conformación tenderá a conducir la conformación hacia la estructura verdadera de la proteína.

La 1UTG o uteroglobina es una proteína de 4 hélices que tiene 70 residuos (aminoácidos). La uteroglobina pertenece a la familia de las secretoglobinas, que son proteínas diméricas secretadas que sólo se encuentran en los mamíferos. La secuencia de esta proteína la hemos obtenido del servidor del PDB. Para las estructuras secundarias usamos las restricciones de estructura supersecundaria. La mejor energía calculada es de 93,7457 $kcal\,mol^{-1}$ la cual nos da un $RMSD_{C_\alpha}$ de 12,917, mientras que el mejor $RMSD_{C_\alpha}$ es de 12,671 con una energía de 96,8755 $kcal\,mol^{-1}$, estos resultados también se encuentran en la tabla 15.

Tabla 15. La mejor energía y el mejor RMSD de 1UTG.

	$Energía\ (kcal\ mol^{-1})$	$RMSD_{C_\alpha}$(Å)
Min energía	93,7457	12,917
Min RMSD	96,8755	12,671

En la ilustración 21 se puede observar la similitud entre la estructura 3D real de la proteína 1UTG (imagen de la derecha) y la predicha gracias a nuestro algoritmo MOABC (imagen de la izquierda).

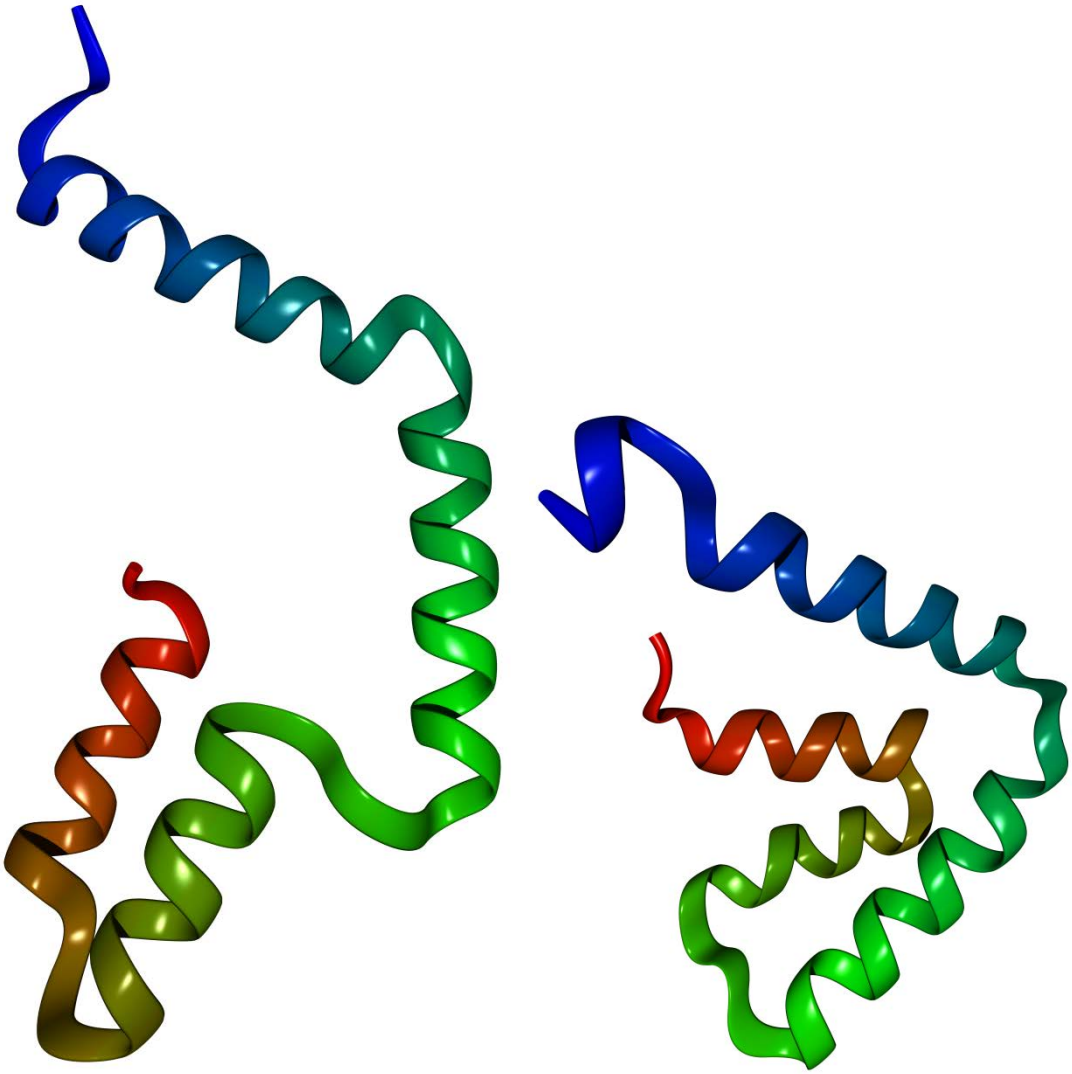

Ilustración 21. Conformaciones (estructura 3D) predicha (a la izquierda) y nativa (a la derecha) para la proteína 1UTG ($RMSD_{C_\alpha}$ = 12,671 Å). Imágenes de elaboración propia gracias al programa BALLView.

La 1CRN o cambrina es una proteína de 46 residuos (aminoácidos) con doble hélice y un par de hebras. Tiene tres enlaces disulfuro, cuyas limitaciones no usamos. La crambina es una pequeña proteína de almacenamiento de semillas de la col. Pertenece a las tioninas y ha sido ampliamente estudiada por cristalografía de rayos X. Para esta proteína, la secuencia también la hemos obtenido del servidor PDB y las estructuras de restricciones que hemos usado en las pruebas son la estructura supersecundaria. La mejor estructura (RMSD) calculada tenía un $RMSD_{C_\alpha}$= 8,001 Å con una energía de 374,9686 $kcal\ mol^{-1}$. En este caso la mejor energía coincide con la energía de la mejor estructura (RMSD), el resultado está recopilado en la tabla 16.

Tabla 16. La mejor energía y el mejor RMSD de 1CRN.

	$Energía\,(kcal\ mol^{-1})$	$RMSD_{C_\alpha}$(Å)
Min energía y RMSD	374,9686	8,001

En la ilustración 22 se puede observar la similitud entre la estructura 3D real de la proteína 1CRN (imagen de la derecha) y la predicha gracias a nuestro algoritmo MOABC (imagen de la izquierda).

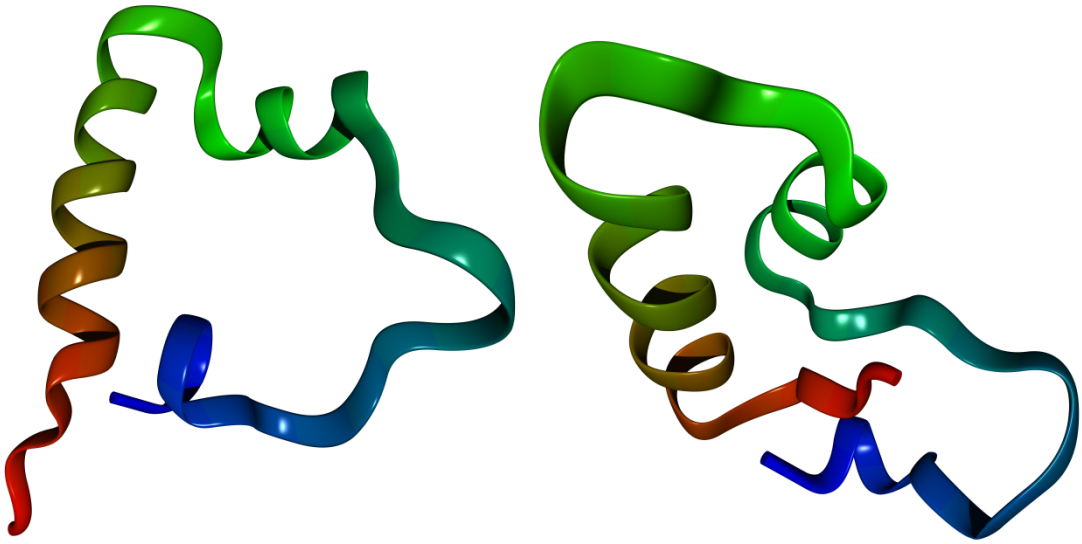

Ilustración 22. Conformaciones (estructura 3D) predicha (a la izquierda) y nativa (a la derecha) para la proteína 1CRN ($RMSD_{C_\alpha}$ = 8,001 Å). Imágenes de elaboración propia gracias al programa BALLView.

13. Comparaciones con otros resultados de la literatura

Compararemos nuestro algoritmo, nuestra versión del MOABC, con otros trabajos de este campo, en particular con el I-PAES (Cutello et al., 2006), uno de los algoritmos de referencia en este área de investigación. Todos los resultados de energía con los que nos comparemos han utilizado la misma función de energía de CHARMM que nosotros. Ambos algoritmos (MOABC e I-PAES) utilizan operadores de mutación de alto nivel, es decir, la proteína se manipula a nivel de aminoácido. Además, ambos algoritmos multiobjetivo han ejecutado el mismo número de evaluaciones (ver apartado 12), para que de esta forma las comparaciones sean justas.

A continuación, compararemos nuestros resultados con los que obtiene el algoritmo I-PAES y los mostraremos en las tablas 17 y 18.

Tabla 17. Comparativa con el I-PAES con las energías mínimas.

	MOABC		**I-PAES**	
	Energía(kcal mol^{-1})	*RMSD(Å)*	*Energía(kcal mol^{-1})*	*RMSD(Å)*
MET	1,8813	1,494	-20,56	1,74
1ZDD	-1122,7198	2,3	-1052,09	2,27
1UTG	93,7457	12,917	573,89	5,31
1CRN	374,9686	8,001	509,09	6,99

Tabla 18. Comparativa con el I-PAES con los mejores RMSD.

	MOABC		**I-PAES**	
	Energía(kcal mol^{-1})	*RMSD(Å)*	*Energía(kcal mol^{-1})*	*RMSD(Å)*
MET	5,1674	1,392	-20,56	1,74
1ZDD	-1063,4068	2,211	-1037,79	2,22
1UTG	96,8755	12,671	1170,85	4,27
1CRN	374,9686	8,001	752,42	4,375

Se puede observar que en ambas tablas en la mayoría de las pruebas (1ZDD, 1CRN y 1UTG), si nos centramos en los resultados multiobjetivo (mínima energía), superamos a los resultados obtenidos por I-PAES, donde la mejora en 1ZDD no es mucha, pero en 1CRN es dos veces mejor y para la 1UTG es mucho mejor. Por otro lado, si comparamos también con los resultados biológicos (RMSD), en este caso, obtenemos resultados malos con 1UTG y 1CRN, pero con MET sí es cierto que obtenemos mejor RMSD a pesar de tener una energía peor y para la 1ZDD más o menos el RMSD es similar, pero como ya hemos comentado antes para esta proteína nosotros obtenemos mejores energías.

14. Conclusiones

En este apartado comentaremos las impresiones que se han obtenido con el trabajo realizado y los resultados obtenidos. Se concluirá el apartado explicando algunas de las posibles ampliaciones que se le podrían realizar a este trabajo para intentar mejorar sus resultados.

14.1. Principales aportaciones

Tras finalizar el trabajo se considera que se han alcanzado todos los objetivos propuestos. El principal objetivo de este trabajo es poder predecir la estructura de las proteínas mediante optimización multiobjetivo. Este objetivo se ha alcanzado aplicando el algoritmo MOABC, la función de energía de CHARMM y el software Tinker. Mediante estos elementos se ha conseguido predecir la estructura terciaria de las proteínas.

Referente a la optimización multiobjetivo, la función energética de CHARMM está compuesta principalmente por dos interacciones, local (energía de enlace) y no local (energía no de enlace) entre los átomos que están en conflicto. Estas dos interacciones energéticas están en conflicto, es decir, si una interacción disminuye la otra aumenta, pero mientras este proceso continúa, hay efectos de compensación que causarán la minimización de la energía total. Aquí es donde ha estado nuestra labor de optimización multiobjetivo, y se puede apreciar que hemos obtenido buenos resultados energéticos.

El siguiente objetivo era estudiar el coste temporal e intentar reducirlo mediante paralelismo. Este otro objetivo también se ha visto cumplido tal y como se explica más detalladamente en el apartado 9 y el apartado 11. Hemos estudiado dónde estaba el mayor gasto temporal y hemos aplicado paralelismo (OpenMP) en esas partes del código; pero no fue suficiente con eso, sino que para poder aplicarlo nos vimos en la necesidad de mediar con las operaciones de E/S para poder paralelizar adecuadamente. Tras solucionar eso conseguimos aplicar el paralelismo y reducir el tiempo de cómputo, que pasó de ser 10 días a llegar nada más que a las 9 horas.

Por último, cabría mencionar las conclusiones que sacamos de nuestros resultados respecto a otros algoritmos. Las conclusiones de las comparativas las dividiríamos en dos partes. En una primera parte, los resultados energéticos de nuestro algoritmo son bastantes positivos, ya que en tres de cuatro proteínas distintas obtenemos mejores resultados energéticos que el I-PAES (uno de los algoritmos de referencia en el campo). Se podría decir que de cara a los valores energéticos nuestro algoritmo es bueno, pero aquí surge la segunda parte de esta conclusión. Los resultados biológicos, es decir, los que nos aportan los RMSD o sea ser la disposición espacial de la proteína, no son tan positivos como los energéticos. Esto quiere decir que a pesar de obtener proteínas con mejores energías no son siempre tan similares espacialmente a la proteína nativa como debería.

En definitiva, se puede decir que el trabajo ha cumplido completamente con los objetivos planteados al inicio de él, tal vez dejando un poco que desear en los resultados biológicos, pero aún así con buenas sensaciones para un trabajo futuro.

14.2. Trabajo futuro

Como trabajo futuro se propone la ampliación de este estudio a más proteínas, que no se ha realizado en este trabajo fin de grado por la enorme cantidad de tiempo que esto supondría. Además, también puede ser interesante diseñar/utilizar otros algoritmos multiobjetivo para compararlos entre sí y ver las diferencias que se producen en los resultados según las características de cada algoritmo. Finalmente, también parece interesante el estudio de otros modelos de energía. Además del modelo de CHARMM también existen otros modelos como el de Amber, MMFF (*Merck Molecular Force Field*), etc. Es muy probable que el uso de distintos modelos de energía dé lugar a distintos resultados a nivel biológico (valores de RMSD, y por tanto, calidad de la estructura 3D predicha para la proteína).

15. Referencias bibliográficas

Calvo J.C. y Ortega J. (2009). *"Parallel Protein Structure Prediction by Multiobjective Optimization"*. 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, February 2009, pp. 268-275, DOI: 10.1109/PDP.2009.13.

Calvo J.C., Ortega J. y Anguita M. (2011a). *"Comparison of Parallel Multi-Objective Approaches to Protein Structure Prediction*". **The Journal of Supercomputing**, November 2011, Volume 58, Issue 2, pp. 253–260, DOI: 10.1007/s11227-009-0368-4.

Calvo J.C., Ortega J. y Anguita M (2011b). *"PITAGORAS-PSP: Including Domain Knowledge in a Multi-Objective Approach for Protein Structure Prediction"*. **Neurocomputing**, May 2011, Volume 74, pp. 2675–2682. DOI: 10.1016/j.neucom.2011.04.003.

Calvo J.C., Ortega J., Anguita M, Urquiza J.M. y Florido J.P. (2009). *"Protein Structure Prediction by Evolutionary Multi-objective Optimization: Search Space Reduction by Using Rotamers"*, IWANN, June 2009, Part I, LNCS 5517, pp. 861–868, DOI: 10.1007/978-3-642-02478-8_108.

Cutello V., Narzisi G. y Nicosia G. (2005). *"A Class of Pareto Archived Evolution Strategy Algorithms Using Immune Inspired Operators for Ab-Initio Protein Structure Prediction"*, EvoWorkshops, March–April 2005, LNCS 3449, pp. 54–63. DOI: 10.1007/978-3-540-32003-6_6.

Cutello V., Narzisi G. y Nicosia G. (2006). *"A Multi-Objective Evolutionary Approach to the Protein Structure Prediction Problem"*, **J. R. Soc. Interface**, February 2006, Volume 3, Issue 6, pp. 139–151, DOI: 10.1098/rsif.2005.0083.

Cutello V., Narzisi G. y Nicosia G. (2008). "*Computational Studies of Peptide and Protein Structure Prediction Problems via Multiobjective Evolutionary Algorithms"*. **Multiobjective Problem Solving from Nature**, part of the Natural Computing Series, 2008, pp. 93-114, DOI: 10.1007/978-3-540-72964-8_5.

Deb K., Pratap A., Agarwal S. y Meyarivan T. (2002) "*A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II*". **IEEE Trans Evol Comput**, April 2002, Volume 6, Issue 2, pp. 182–197, DOI: 10.1109/4235.996017.

Del Pozo Fernández, R. (2011). "*Apuntes de Segundo de Bachillerato de Biología, 1ª Evaluación"*. Colegio Diocesano San Atón, Badajoz.

Glover F. (1986). "*Future Paths for Integer Programming and Links to Artificial Intelligence*". **Computers and Operations Research**, 1986, Volume 13, Issue 5, pp. 533–549, DOI: 10.1016/0305-0548(86)90048-1.

González Álvarez D.L. (2013). "*Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN"*. Tesis Doctoral, June 2013, Universidad de Extremadura.

Handl J., Kell D.B. y Knowles J. (2007). "Multiobjective Optimization in Bioinformatics and Computational Biology". **IEEE/ACM Transactions on Computational Biology and Bioinformatics**, April 2007, Volume 4, Issue 2, pp. 279–292, DOI: 10.1109/TCBB.2007.070203.

Karaboga D. (2005). "*Artificial Bee Colony (ABC) Algorithm"*. Erciyes University, Turquía, [consulta: febrero 2017], disponible en: http://mf.erciyes.edu.tr/abc/.

Krink T. (2001). *"Swarm Intelligence – Introduction"*. University of Aarhus, Dinamarca, [consulta: febrero 2017], disponible en: http://staff.washington.edu/paymana/swarm/krink_01.pdf.

McLachlan, A.D. (1982). "*Rapid Comparison of Protein Structures*". **Acta Cryst**, 1982, Volume A38, pp. 871-873, DOI; 10.1107/S0567739482001806.

Muñoz M.A., López J.A. y Caicedo E.F. (2008). "*Inteligencia de Enjambres: Sociedades para la Solución de Problemas*". **Ingeniería e Investigación**, August 2008, Volume 28, Issue 2, pp. 119-130.

Rubio Largo, A. (2013). "*Multiobjective Metaheuristics and Parallel Computing for Optimizing WDM Optical Networks"*. Tesis Doctoral, July 2013, Universidad de Extremadura.

Varios autores (2004). *"Algoritmos Evolutivos y Meméticos"*. Curso de Postgrado – UC3M [consulta: febrero 2017], disponible en: http://www.exa.unicen.edu.ar/escuelapav/cursos/bio/l2.pdf.

Varios autores (2017). *"Algoritmo Genético"*. Wikipedia: la enciclopedia libre, [consulta: febrero 2017], disponible en: https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico

Varios autores (2017). *"Algoritmo Evolutivo"*. Wikipedia: la enciclopedia libre, [consulta: febrero 2017], disponible en: https://es.wikipedia.org/wiki/Algoritmo_evolutivo

Varios autores (2017). *"Evolutionary Algorithm"*. Wikipedia: la enciclopedia libre, [consulta: febrero 2017], disponible en: https://en.wikipedia.org/wiki/Evolutionary_algorithm

Varios autores (2017). "*Swarm Intelligence*". Wikipedia: la enciclopedia libre, [consulta: febrero 2017], disponible en: https://en.wikipedia.org/wiki/Swarm_intelligence

Yang X.S. (2009). "*Firefly Algorithms for Multimodal Optimization*", 5th International Symposium of Stochastic Algorithms: Foundations and Applications (SAGA'09), October 2009, LNCS 5792, pp. 169–178, DOI: 10.1007/978-3-642-04944-6_14.