UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

TRATAMIENTOS MULTIOBJETIVOS APLICADOS AL PROBLEMA DE PLEGAMIENTO DE PROTEÍNAS BAJO EL MODELO *HP*.

TESIS

PARA OBTENER EL GRADO DE:

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

PRESENTA

ING. MANUEL SERVANDO FLORES CHACÓN

DIRECTOR DE TESIS

DR. LUIS CARLOS GONZÁLEZ GURROLA

CHIHUAHUA, CHIHUAHUA. NOVIEMBRE DE 2016

UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

TRATAMIENTOS MULTIOBJETIVOS APLICADOS AL PROBLEMA DE PLEGAMIENTO DE PROTEÍNAS BAJO EL MODELO *HP*.

TESIS

PARA OBTENER EL GRADO DE:

MAESTRÍA EN INGENIERÍA EN COMPUTACIÓN

APROBADO:

Dr. Luis Carlos González Gurrola

Graciela Ramirez A.

Dra. Graciela Maria de Jesús Ramírez Alonso

Dr. Fernando Martínez Reyes

NOVIEMBRE DE 2016

CHIHUAHUA, CHIHUAHUA.

UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

Facultad de Ingeniería

24 de noviembre de 2016

ING. MANUEL SERVANDO FLORES CHACÓN

Presente

En atención a su solicitud relativa al trabajo de tesis para obtener el grado de Maestro en Ingeniería, nos es grato transcribirle el tema aprobado por esta Dirección, propuesto y dirigido por el director **Dr. Luis Carlos González Gurrola** para que lo desarrolle como tesis, con el título: **"TRATAMIENTOS MULTIOBJETIVO APLICADOS AL PROBLEMA DE PLEGAMIENTO DE PROTEÍNAS BAJO EL MODELO HP".**

ÍNDICE

Facultad de Ingeniería
Circuito No.1, Campus Universitario 2
Chihuahua, Chih. C.P. 31125
Tel. (614) 442-95-00 www.fing.uach.mx

UNIVERSIDAD INTERNACIONAL

UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

Facultad de Ingeniería

3. Metodología

3.1. NSGA-II

3.1.1. Representación del cromosoma

3.1.2. Operador de cruzamiento

3.1.3. Mutación multigen

3.1.4. Mutación Pull-Moves Neighbors

3.1.5. Variable Neighborhood Search

3.1.6. Selección de Padres

3.1.7. Selección de supervivientes y mecanismo de diversidad

4. Resultados

4.1. Comparación entre las funciones de energía multi-objetivo

4.2. Resultados 2D

4.3. Resultados 3D

5. Conclusiones

Referencias

Solicitamos a Usted tomar nota de que el título del trabajo se imprima en lugar visible de los ejemplares de las tesis.

A T E N T A M E N T E
"naturam subiecit aliis"

EL DIRECTOR

M.I. JAVIER GONZÁLEZ CANTÚ

FACULTAD DE INGENIERÍA U.A.CH. CHIHUAHUA Facultad de Ingeniería DIRECCIÓN

EL SECRETARIO DE INVESTIGACIÓN Y POSGRADO

DR. FERNANDO RAFAEL ASTORGA BUSTILLOS

Abstract

Proteins are linear sequences of amino acids, these must be able to fold correctly without more information than the coding of amino acids that comprise it and the interactions that are established between them, under certain physiological conditions, the proteins are folded up to find its native three-dimensional conformation, Or in other words the functionally active conformation. The hydrophobic-polar model is an abstraction of the behavior of proteins, based on the hydrophobic iterations of certain amino acids. The model looks for a function of mono-objective energy to determine the functional protein configuration.

In the literature there are different versions of the energy function for the model, which approach the problem from different points of view. Within these functions are multi-objective alternatives, which seek to abound over the target space in the hope that such search will provide better results than the original energy function. In this work we present aspects to work with a multi-objective version of the energy function used by the model, using a multi-objective evolutionary algorithm. The experiments showed that the proposed algorithm can be used with excitation for instances of less than or equal length 36 for a square grid in 2D, whereas for a 3D the algorithm allowed to have favorable results for instances less than or equal to 18. By subjecting the algorithm to larger instances, the algorithm shows competitive results with respect to those shown in the literature.

Resumen

Las proteínas son secuencias lineales de aminoácidos, estas deben ser capaces de plegarse correctamente sin más información que la codificación de aminoácidos que la conforman y las interacciones que se establecen entre ellos, bajo ciertas condiciones fisiológicas, las proteínas se pliegan hasta encontrar su conformación tridimensional nativa, o en otras palabras la conformación funcionalmente activa. El modelo Hidrofóbico-polar, es una abstracción del comportamiento de las proteínas, basado en las iteraciones hidrofobias que presentan ciertos aminoácidos. El modelo busca mediante una función de energía mono-objetivo determinar cuál es la configuración proteica funcional.

En la literatura existen diferentes versiones de la función de energía para el modelo, las cuales abordan el problema desde diferentes puntos de vista. Dentro de estas funciones se encuentran alternativas multi-objetivo, que buscan abundar sobre el espacio objetivo con la esperanza de que dicha búsqueda proporciones mejores resultados que la función de energía original. En este trabajo se presentan aspectos para trabajar con una versión multi-objetivo de la función de energía que utiliza el modelo, utilizando un algoritmo evolutivo multi-objetivo. Los experimentos realizados, mostraron que el algoritmo propuesto, puede utilizarse con excitó para instancias de longitud menor o igual 36 para una retícula cuadrada en 2D, mientras que para una en 3D el algoritmo permitió tener resultados favorables para instancias menor o igual 18. No obstante al someter al algoritmo a instancias más grandes, el algoritmo muestra resultados competitivos con respecto a los mostrados en la literatura.

Dedicatoria

A mi familia por todo el apoyo que siempre me ha brindado durante mi formación personal y profesional.

Agradecimientos

Primeramente agradezco a Dios por haberme puesto en este camino, por la salud que me ha permitido tener durante todo este tiempo para así poder realizar mis estudios de posgrado de la mejor forma. Agradezco al

Consejo Nacional de Ciencia y Tecnología (Conacyt), por su apoyo económico brindado para la realización de mis estudios de posgrado.

A la Universidad Autónoma de Chihuahua y en especial a la Facultad de Ingeniería, por haberme dado la oportunidad de estudiar esta maestría y contribuir en gran medida a mi formación académica y personal.

A mi director de Tesis, el Dr. Luis Carlos González Gurrola, por haberme adoptado como tesista durante todo este tiempo. Por todo el apoyo brindado durante la realización del proyecto y durante todo mi camino por posgrado. Por su confianza, paciencia y todo su conocimiento compartido.

De la misma manera agradezco a *CINVESTAV* y en especial al Dr. Gregorio Toscano Pulido, por haberme guiado en mi estancia de investigación.

Por último pero no menos importante a mi familia y a mis amigos por siempre apoyarme en todo momento. Sin su gran apoyo esto no hubiera sido posible.

Índice de Contenido

Índice de Contenido

Índice de figuras

Índice de tablas

1

Introducción.

La Bioinformática es la disciplina científica que combina biología y ciencias computacionales para organizar y analizar información biológica (secuencias de genomas, estructuras de proteínas, etc), con la finalidad de ayudar a la resolución de problemas de la biología molecular, con la ambición de mejorar la condición y calidad de vida, tratamiento de enfermedades, la producción de alimentos sólo por mencionar algunas tareas [42]. Por otra parte, dentro de las ciencias computacionales, la Bioinformática contiene problemas de gran complejidad, que usualmente son problemas *NP-Complete*, lo que motiva a usar técnicas heurísticas, con el objetivo proporcionar soluciones aproximadas.

Dentro de la biología, encontramos a las proteínas, moléculas que desempeñan una gran cantidad de funciones en las células de los seres vivos. Las proteínas son cadenas lineales de bloques llamados aminoácidos, que bajo ciertas condiciones físicas se pliegan para formar complejas estructuras tridimensionales [54, 50, 23]. En condiciones fisiológicas la estructura tridimensional se conoce como estructura nativa, y se considera como la estructura termodinámica más estable (menor energía global) de las combinaciones posibles. A esta configuración es la que se denomina funcionalmente activa.

Actualmente la suposición más clara que se tiene sobre el plegamiento de las proteínas, señala que la información necesaria para determinar un buen plegamiento, está contenida en la estructura primaria (estructura más básica que puede tener una proteína), o que es lo mismo, la secuencia de aminoácidos y las interacciones que ocurren entre ellos. Cómo una proteína se pliega hasta alcanzar su estructura nativa, es uno de los grandes desafíos que enfrenta la biología molecular. A este problema se le conoce como el problema de predicción de estructura de la proteína (PSP) [17] y representa una de las áreas de investigación más activas y desafiantes en el campo de la bioinformática.

Debido a la cantidad exponencial de conformaciones que una proteína real puede adquirir, resulta difícil hacer búsquedas exhaustivas para encontrar su estructura nativa. Por esta razón han surgido modelos

que simplifican la simulación del plegado de proteínas y ayudan a entender de manera esencial este proceso [34]. Uno de estos, es el modelo hidrofóbico-polar (*HP*), el cual está basado en las observaciones de proteínas reales, donde se ha encontrado que la fuerza que dirija el plegado de la proteína se basa en el efecto hidrofobia.

Ken A. Dill. [17] propusieron el modelo *Hydrophobic-Polar* (*HP*), actualmente uno de los modelos más usados para modelar proteínas [37][47]
[23]. En su trabajo Dill menciona cómo una proteína se puede abstraer en una representación más sencilla de dos dimensiones, para esto, explica que el modelo agrupa los aminoácidos que componen las proteínas en dos clases: los residuos (aminoácidos en la cadena) hidrofóbicos y residuos polares (hidrófilos). El modelo aprovecha el comportamiento de los aminoácidos hidrofóbicos los cuales tienden a colocarse en el centro de la proteína, tratando de mantenerse juntos para protegerse y alejarse del agua, mientras tanto los residuos polares interactúan mejor con el agua por lo que causa que se desplacen hacia el borde de la estructura. Contemplando este fenómeno el modelo *HP*, pretende simular la estructura nativa (funcional), de las proteínas. Si bien es cierto que el modelo *HP* reduce la complejidad del problema, la búsqueda de la estructura nativa, el problema de plegado de proteínas bajo este modelo sigue siendo aun un problema *NP-Complete* [4, 12]. En la literatura encontramos una gran variedad de estrategias computacionales que han sido empleadas para simular y aplicar el modelo *HP*, con la intención de reducir el tiempo y mejorar el rendimiento de la búsqueda hacia la conformación nativa.

En 2002, Krasnogor et al. [36] utilizaron un algoritmo memético, con una serie de modificaciones propias por los autores, al cual llamaron algoritmo *Multimeme*, en esta nueva versión del memético, los individuos aprenden de forma individual el método de búsqueda local (meme) que va a emplear. De esta forma los individuos quedan conformados por su material genético más un alelo que indica el meme a utilizar. En este trabajo el algoritmo multimeme fue puesto a trabajar sobre cuatro diferentes modelos: modelo *HP* en un cuadricula cuadrada, *HP* en un cuadricula triangular, “Functional Model Proteins” en una cuadricula cuadrada y en una cuadricula diamante. Los autores compararon su propuesta con uno de los primeros algorimtos en tratar el PSP bajo el modelo *HP* , el algoritmo *PERM* (*pruned-enriched Rosenbluth method*) [2], en diferentes instancias del modelo funcional. A partir de su experimentación, encontraron que la calidad de las soluciones encontradas por el algoritmo Multimeme fueron mejor o al menos similares a las soluciones encontradas por PERM en la mayoría de las instancias a prueba.

Entre los métodos para resolver el problema de plegamiento de proteínas encontramos el algoritmo de la colonia de hormigas (*Ant Colony Optimization*, ACO) [47] utilizado por Shmygelska y Hoos. Su propuesta sobre este algoritmo posee la capacidad de trabajar con el modelo *HP* en 2 y 3 dimensiones, sin mencionar

que utilizaba mecanismos propios de un algoritmo Monte Carlo para enriquecer sus agentes. En cuanto sus resultados los autores concuerdan que realizar una comparativa con otros métodos en la literatura es una tarea compleja, pues aclaran que la mayoría de éstos trabajos reportan el número de conformaciones válidas encontradas, mientras que ellos sólo manejan el tiempo computacional requerido para llegar a su solución. No obstante el valor de la función objetivo para las soluciones es suficiente para confirmar que ACO logra encontrar soluciones lo suficientemente aceptables para las instancias con las que se trabajó.

Si bien, abstraer comportamientos biológicos para tratar de resolver problemas NP, no es nada nuevo, existen muchos algoritmos que han comprobado su eficiencia a la hora de realizar sus tareas. Tal es el caso de los algoritmos basados en el sistema inmune de los animales vertebrados (AISs, Artificial Immune System). Cutello et al. [13] presentan un algoritmo inmune o IA, que al igual que la mayoría de los trabajos relacionados al tema, trabaja con el modelo *HP* sobre cuadrillas para 2 y 3 dimensiones. El IA presentado en ese trabajo asemeja el comportamiento de los linfocitos o células *B*, las cuales tienen la peculiaridad de clonarse una vez que han recopilado y memorizado la información de los antígenos que invaden al organismo. Dentro del algoritmo las células *B* representan soluciones potenciales que poseen la capacidad de clonarse a sí mismas una *n* cantidad de veces, las cuales mutan con la esperanza de converger a la conformación nativa, por otra parte el algoritmo depende de la duración de vida que el usuario defina para las células. Si bien los autores establecen que la esperanza de vida que ellos plantean funciona para la mayoría de las instancias de prueba, sus resultados no son sobresalientes para las instancias de proteínas con más de 35 aminoácidos en la cadena.

Siguiendo con la idea de encontrar la conformación nativa con base en tener una población de soluciones que surgieron de la clonación de un de las soluciones, Thachuk et al. [49] reportan en su trabajo, el algoritmo *"Replica Exchange Monte Carlo"* (REMC). El cual toma una posible solución, la cual es reproducida una *x* cantidad de veces, con la finalidad de perturbar dichas replicas, utilizando diferentes métodos [52, 38] para mejorar la solución. Sus resultados en comparación con el *ACO* de Shmygelska mejoraron no sólo en rendimiento, sino también encontrando nuevos óptimos conocidos para el *PSP*. Su trabajo establece también la clara superioridad del método *"Pull-Moves"*como mutación para cuadrillas representadas bajo el modelo *HP*.

Dentro de los múltiples métodos que se han utilizado para enfrentar al PSP, encontramos los algoritmos basados en la estimación de distribución (*"Estimation of Distribution Algorithm"*, EDA), metaheurísticas derivadas de los algoritmos evolutivos, que en lugar de buscar la solución de un problema, los EDA aprenden y aprovechan las regularidades del espacio de búsqueda en forma de dependencias probabilistas. Santana

et al. [46] propusieron en su trabajo tres variantes, las cuales trabajan cada una con un modelo probabilístico para estimar la distribución de probabilidad de las variables del modelo. Dichos modelos utilizados fueron: *Modelo Ocultos de Markov*, *Modelo Basado en Árboles* y el último una mezcla de árboles (*Mixture of Trees*). Aunque igual que la mayoría de los métodos, el algoritmo encuentra fácilmente las soluciones para instancias pequeñas y para las más grandes se queda a uno o dos pasos del óptimo conocido.

Dentro de está área podemos encontrar comúnmente algoritmos híbridos, formados por la combinación entre 2 o más metaheurísticas. Chira et al. [8] propusieron un algoritmo evolutivo hibridado con una búsqueda local *"hill-climbing"* y un método de diversidad. Esta nueva propuesta *"ELF"* (*Evolutionary algorithm hybridized with Local hill-climbing search and Fingerprint-diversification.*), utiliza un método de recombinación y uno de mutación propios de un algoritmo evolutivo, con la diferencia de que estos generan k ($k>2$) descendientes y selecciona m mejores para continuar con el ciclo del algoritmo evolutivo. Por otra parte, se implementó un mecanismo para garantizar la diversidad de la población, el cual calcula una huella digital única para cada individuo. El mecanismo opera comparando las huellas de todos los individuos en la población, generando una métrica de qué tan parecidos o diferentes son, si la métrica determina que los individuos son muy parecidos, el método reemplaza I individuos con nuevos generados de forma aleatoria. Sobre los resultados mostrados, los autores aclaman encontrar el óptimo conocido para las instancias estándar para el modelo *HP*, en su versión de dos dimensiones. Aunque al igual que las mayorías de los trabajos en la literatura se le dificulta encontrar el mejor para las instancias de mayor tamaño, manteniendo un porcentaje de ocurrencia bajo para dichas instancias de proteínas.

Hasta el momento todos los trabajos manejan la versión en dos dimensiones del modelo *HP* y sólo algunos incluyen la versión en 3D para éste. Lin et al. [39] proponen utilizar únicamente el modelo *HP* en 3D dimensiones puesto que consideran que dado el hecho de lo que el problema PSP busca la estructura terciaria nativa, esta versión se asemeja más a la realidad de las proteínas. Por lo demás los resultados mostrados por los autores sobre las instancias estándar del modelo, fueron puestas a prueba sobre la metaheurística *Optimización por Nube de Partículas* (*PSO*). No obstante el PSO continúa con los mismos problemas presentados por la mayoría de los algoritmos, al exponer porcentajes bajos de ocurrencia en óptimos, para las instancias de mayor tamaño.

Uno de los métodos más conocidos para atacar al problema PSP bajo el modelo *HP* es sin duda alguna el método de *"Pull-Moves"* [38]. Rego et al. [43] propusieron un algoritmo, que con base en alteraciones de una solución actual, genera de forma dinámica un conjunto de soluciones que se adaptan conforme el algoritmo avanza. El *"Filter and Fan" (F&F)* utiliza un solución inicial cómo la base de un árbol el cual va generando ramas a través de aplicar el método *Pull-Moves* a ciertos nodos, con la esperanza de encontrar una

mejor solución en la cima del árbol. El *F&F* ha demostrado encontrar el óptimo conocido para cada una de las instancias base del modelo *HP*, pero con la contra parte de que el algoritmo requiere una gran cantidad de recursos para hacer sus cálculos. Además de trabajar únicamente en 2D.

Uno de los trabajos más recientes que habla sobre el *PSP*, es el trabajo presentado por Boskovic et al. [6], ellos propusieron un algoritmo genético, con la peculiaridad de que éste implementa un mecanismo para dividir la población entre diferentes clúster, poniendo los individuos con características en el fenotipo similares, en un mismo grupo. Un vez hecho esto, el algoritmo continúa cómo un algoritmo genético normal, agregando un mecanismo de búsqueda local al fin de cada generación. Al finalizar las generaciones, el proceso de agrupación vuelve a realizarse. Los resultados mostrados por los autores se efectuaron sobre las instancias base del modelo *HP* y las instancias "Harvard" en 3 dimensiones reportando buenas soluciones para la mayoría de las instancias trabajadas.

1.1. Funciones de energía para el modelo HP

Para predecir la conformación nativa, se requiere calcular una función de energía, bautizada cómo función de Dill [23], la cual sólo toma las interacciones entre aminoácidos hidrofóbicos adyacentes en el espacio, pero no consecutivos en la secuencia, tal y como se ven en la Figura 1.1, la proteína representada bajo el modelo *HP* contiene 10 aminoácidos polares y 10 hidrofóbicos, entre los cuales existen 9 interacciones representadas por las lineas punteadas. Los residuos hidrofóbicos que están adyacentes en el espacio pero no adyacentes en la secuencia (estructura más básica que puede tener una proteína) incrementan un factor negativo constante (-1). El estado nativo entonces puede ser considerado como el de energía global mínima.

Dado que buscar la conformación nativa representa un problema de optimización combinatoria complejo, se ha observado que la función *Dill*, proporciona una pobre discriminación entre posibles soluciones [23], lo que habilita a que posibles soluciones con capacidad de mejorar, sean despreciadas. Debido a esto, distintos autores han presentado una alternativa para la función de energía de Dill, con el fin de mejorar el rendimiento de las metaheurísticas aplicadas, y poder presenciar un nivel más alto de discriminación entre posibles conformaciones y además mejorar el rendimiento del algoritmo con el que se trabaja.

Krasnogor N. et al. en [37], presentan una modificación a la función de energía de Dill, en la cual no sólo considera los contactos hidrofóbicos en la secuencia de aminoácidos, si no también lo compacto que una conformación pude ser. Esto con el fin de determinar qué conformación es mejor cuando se tienen algunas con el mismo valor de energía. En la Figura 1.2 se puede observar cómo la conformación *a* es más compacta que la *b*, por lo tanto de acuerdo a esta alternativa para la función de energía, los autores proclaman que la

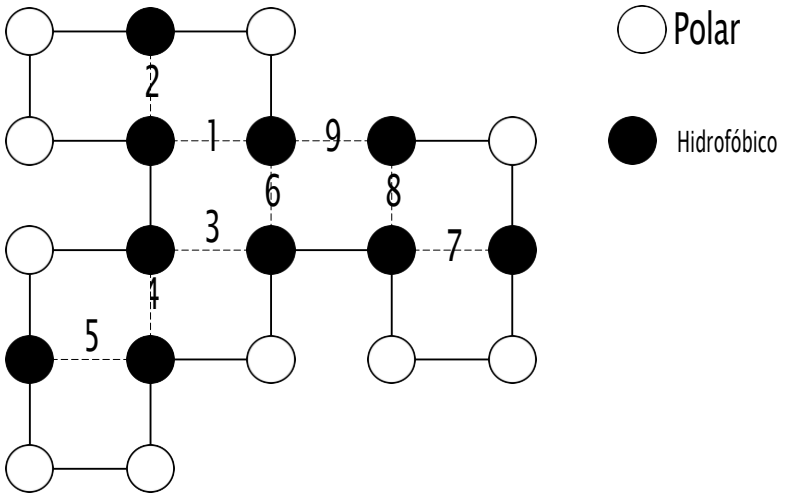

Figura 1.1: Representación del modelo *HP*, con un valor de función de -9.

Figura basada de [23]

estructura *a* nos proporciona un mejor camino hacia la estructura nativa. Usando un algoritmo genético, los autores no logran demostrar una clara evidencia de que su planteamiento haga una mejora altamente notable con respecto a la función original.

Otro de los trabajos, donde se probó mejorar la función de energía de Dill fue presentado por Md. Kamrul Islam et al. En [32], se proponen dos nuevas métricas para incorporar a la función original de Dill, las cuales consistían en encerrar a la estructura en un hipotético cuadrado, y: *a)* medir la cercanía de todos los aminoácidos hidrofóbicos a un hipotético centro y, *b)* medir la cercanía de los aminoácidos polares al perímetro del hipotético cuadrado. Todo esto bajo un algoritmo memético, las métricas implementadas por Islam tienen el objetivo de guiar y mejorar la búsqueda hacia el óptimo para las instancias estándar del modelo *HP*.
Lamentablemente ninguno de los autores presentaba resultados que indicaran que efectivamente existe una mejora al utilizar una función alterna a la función de energía de Dill.
No fue hasta que Mario Garza et al. propone en [23], la implementación de estas versiones alternas, agregando además otras funciones recopiladas de la literatura. Aplicándolas sobre un mismo algoritmo, el cual es en este caso un algoritmo *memético*, con el fin de comprobar cuál de estas funciones lograban hacer una mejor discriminación entre las posibles soluciones. sus resultandos muestran que las funciones alternas de energía con más alto nivel para discriminar fue la función de energía propuestas por Berenboy y avigal [3], seguidas por las funciones: Krasnogor [37] e Islam [32]. En un segundo experimento, los autores comprueban que las funciones de energía Krasnogor e Islam, mejoran el rendimiento del algoritmo memético, mientras que las

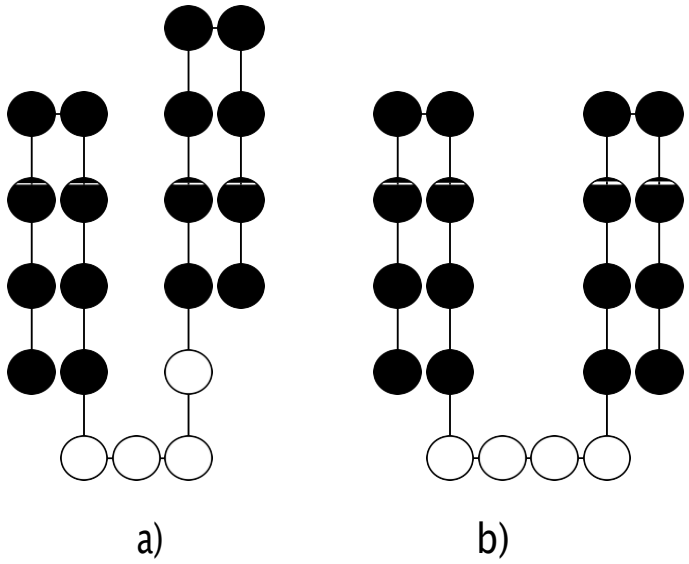

Figura 1.2: Dos conformaciones con igual valor de energía, la estructura en la figura *a* posee menos energía de acuerdo a la nueva formulación de energía.
Figura basada de [37]

funciones de Berenboy y avigal [3] tuvieron el peor rendimiento haciendo la búsqueda más lenta. Aun así, los autores demuestran que sobre ambos criterios la función de energía presentada por *Dill*, es la función de energía que menos información aporta a la búsqueda de la conformación nativa.

Uno de los planteamientos que surgieron para enfrentar el problema de la pobre discriminación de la función de energía de *Dill*, fue propuesto en [25]. La idea consiste en hacer una reformulación de la función *mono-objetivo* propuesta por *Dill*, y transformarla en una función *multi-objetivo*. El vector de funciones creado a partir de la función de energía de Dill, fue propuesto utilizando el método de descomposición. La nueva función *"Parity Descomposition"*, establece 2 diferentes cuentas para cada uno de los objetivos, la primera calcula los contactos de los aminoácidos que están en una posición prima en la secuencia primaria de la proteína, y en cambio la segunda función calcula los contactos de los aminoácidos que son pares en la secuencia primaria. Dicha nueva función *multi-objetivo* fue probada sobre un algoritmo evolutivo *1+1* y una versión de éste llamado *1+1 archive*, en donde los resultados mostrados son alentadores, y logra su propósito de mejorar la discriminación entre soluciones candidatas, de lo que la función de energía de Dill lo hace.

En un segundo trabajo [27], presentan una nueva forma de descomponer la función de energía de Dill, aquí presentan cómo la función de energía *"Locality Descomposition"*, nueva presentación de la función de energía original, costa sólo de verificar la posición en la secuencia de los contactos hidrofóbicos. Por lo tanto el vector de funciones queda establecido de la siguiente manera: la primera función se reduce en un factor

negativo si la resta de las posiciones es menor a un valor de localidad δ, la segunda si la resta es mayor a un valor δ. Con respecto a los resultados los autores aplicaron su nueva propuesta sobre un algoritmo evolutivo *1+1*, y demuestran que este tipo de transformación de mono-objetivo a multi-objetivo tiene la capacidad de discriminar de mejor forma que la función original de Dill e inclusive mejor que la función multi-objetivo *"Parity Descomposition"*.

Otro de los trabajos propuestos donde la función de energía de Dill fue trasformada a multi-objetivo, se formulo en [24]. Para este nuevo esquema primeramente se requiere que los aminoácidos dentro de la secuencia primaria de la proteína se dividan en dos diferentes grupos, establecido esto, la función *"H-subset Decomposition"* puede ser aplicada, en este caso se tiene un vector de 2 funciones. La primera de éstas realiza una sumatoria de las iteraciones entre aminoácidos que se encuentren en un mismo grupo, mientras tanto la segunda función contabiliza las iteraciones que se presentas entre los aminoácidos de diferentes grupos. De igual manera que en los trabajos previos se utilizó un algoritmo genético *1+1* para poner a prueba la nueva reformulación. Encontrando entonces dentro de los resultados que la función logra hacer mejor su tarea que la función de *"Locality Descomposition"*.

El problema de plegado de proteína, específicamente bajo el modelo de Dill, se ha estado abordando ya desde hace tiempo. En la Figura 1.3 se puede apreciar cómo han evolucionado los diferentes enfoques desde que el modelo fue planteado.

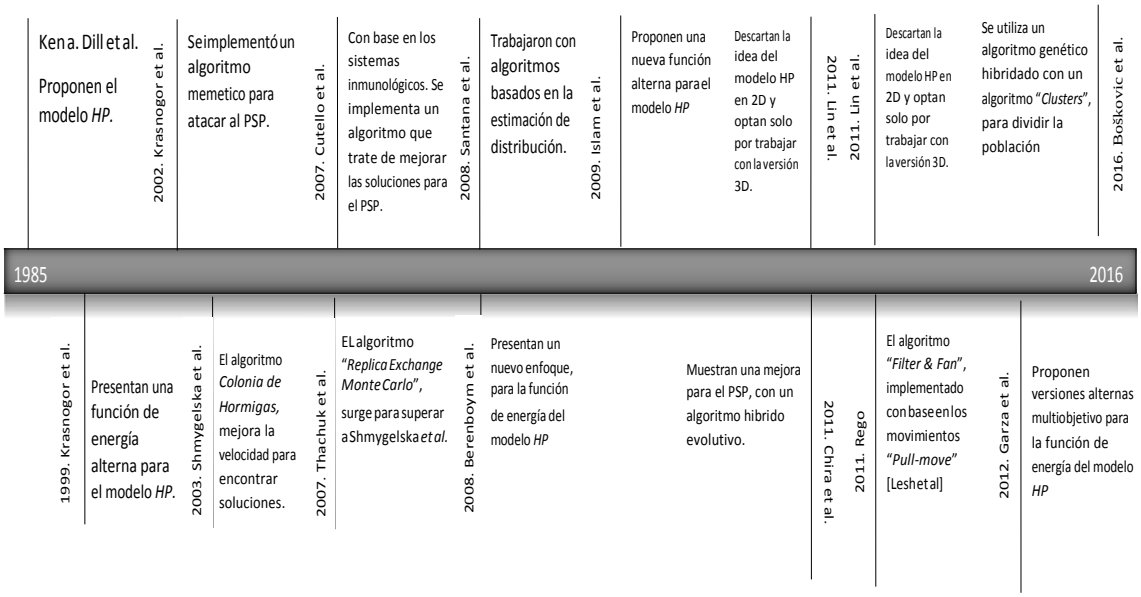

Figura 1.3: Línea del tiempo

Como se ha dicho, distintos trabajos y metaheurísticas se han empleado al PSP, aun así dentro de estos trabajos una idea se ha mantenido constante, y es el hecho de que la función de energía de Dill realiza una pobre discriminación entre las posibles soluciones. Por eso, es claro el hecho, de que existan diferentes versiones para esta función que tratan de realizar una mejora donde la original simplemente falla. Como se ha visto, dentro de dichas versiones se encuentren alternativas que fueron planteadas como funciones multi-objetivo con la finalidad de hacer una exploración de soluciones de una manera diferente, de esta forma, se encontró que los resultados de este enfoque son prometedores, sin embargo no se encontraban a la altura de los resultados en la literatura.

No obstante, la mayoría de estos enfoques multi-objetivo fueron probados sobre metaheurísticas diseñadas originalmente para funciones mono-objetivo, lo cual provoca que exista una duda sobre la verdadera eficacia que puedan llegar a tener dichas funciones o simplemente el enfoque no es una camino factible por el cual ir.

Por lo tanto, con la interrogante acerca del multi-objetivo, lo que se propone hacer para está investigación, es someter el enfoque multi-objetivo, a una metaheurística que este diseñada específicamente para tratar con este tipo de problemas, pues se tiene la esperanza de que el trabajo mutuo entre ambas función y metaheurística, logre que el enfoque multi-objetivo, mejore su comportamiento y sus resultados sean competentes con los trabajos relacionados

1.2. Objetivos.

1.2.1. Objetivo general.

Evaluar una estrategia multi-objetivo aplicada al problema de plegamiento de proteínas, bajo un algoritmo especializado para este tipo deproblemas.

1.2.2. Objetivos específicos.

- Analizar el comportamiento de las funciones propuestas en la literatura.
- Desarrollar un algoritmo evolutivo, apto para el problema.
- Competir con el grado de discriminación de las funciones multi-objetivo propuestas en la literatura.

1.3. Preguntas de investigación.

- ¿Un algoritmo multi-objetivo (MOA, por sus siglas en inglés), tendrá la capacidad de competir contra opciones mono-objetivo para el problema PSP?
- ¿Entre las Funciones multi-objetivo propuestas por Garza et al. cuál se debe usar par en el MOA?
- ¿Qué factor juega la diversidad para la convergencia del MOA para el PSP?

2

Marco Teórico

2.1. Modelo *HP*

Como se mencionó anteriormente, el modelo *HP* es uno de los modelos más estudiados para el plegamiento de proteínas. Desde que fue propuesto en [17], ha tenido la tarea de simplificar el comportamiento de las proteínas al momento de plegarse. Uno de los principales intereses del modelo es explorar de forma completa la naturaleza del *espacio de conformaciones* y *el espacio de secuencias* de las proteínas. Para entender estos términos lo que primero se debe comprender es que el modelo percibe una proteína como una cadena lineal de *n* aminoácidos, donde cada aminoácido puede ser clasificado en dos tipos: *H*, si se trata de un hidrofóbico y *P* en cuyo caso sea polar. Una vez determinado a qué clase corresponde cada uno, la cadena puede ser representada como un camino que evita atravesarse a sí mismo, en una retícula cuadrada en dos dimensiones o modelada de forma cúbica en tres dimensiones, la cual sólo sirve para discretizar el espacio de conformaciones.

Por consiguiente, se define al ***espacio de conformaciones*** como al conjunto de todas las posibles estructuras que una cadena de aminoácidos pueda formar. Debido a todos los diferentes enlaces que una molécula puede realizar, el espacio de conformaciones puede llegar a ser inmenso y buscar en él requeriría un gran tiempo. Con el modelo *HP* se puede discretizar el espacio de conformaciones para realizar una búsqueda relativamente más simple. Dicha discretización como se mencionó anteriormente queda determinada por la retícula que se use. E.g. por cada punto en una retícula cuadrada de dos dimensiones, existen $z = 4$ direcciones hacia donde moverse y por cada punto interno $z - 1 = 3$. Suponiendo una cadena de $n = 10$, existen un aproximado de soluciones de $(z-1)^{(n-1)}$ o lo que es igual a 2034 posibles conformaciones.

Por otro lado el ***espacio de secuencias***, es el conjunto de todas las posibles secuencias de *H* y *P* aminoácidos. Para un modelo en 2D, el espacio de secuencias es: 2^n. E.g. De igual forma si consideramos una cadena de

10 aminoácidos existen 1024 secuencias diferentes en una retícula cuadrada de dos dimensiones.

Como principal objetivo, el modelo pretende calcular las propiedades de ambos espacios, con el fin de encontrar aquellas conformaciones con la mínima energía libre de *Gibbs* [7, 21], o lo que es igual el *estado nativo* (proteína funcionalmente activa). Para determinar cuál es la conformación nativa, recordemos que existen 20 aminoácidos, los cuales pueden ser divididos en dos clases. Gracias a esto podemos reducir nuestro alfabeto de 20 a 2 caracteres y representar las proteínas como una secuencia de caracteres formados por el alfabeto $\{H, P\}$. Con el fin de resaltar los aminoácidos hidrofóbicos, pues se cree que la hidrofobia es uno de los principales factores que determinan como una secuencia puede plegarse hasta la funcionalmente activa [16, 31]. Entretanto a cada conformación para una mismas secuenciara proteica se le asocia un valor de energía *E*, tal y como se muestra en la Figura 2.1.

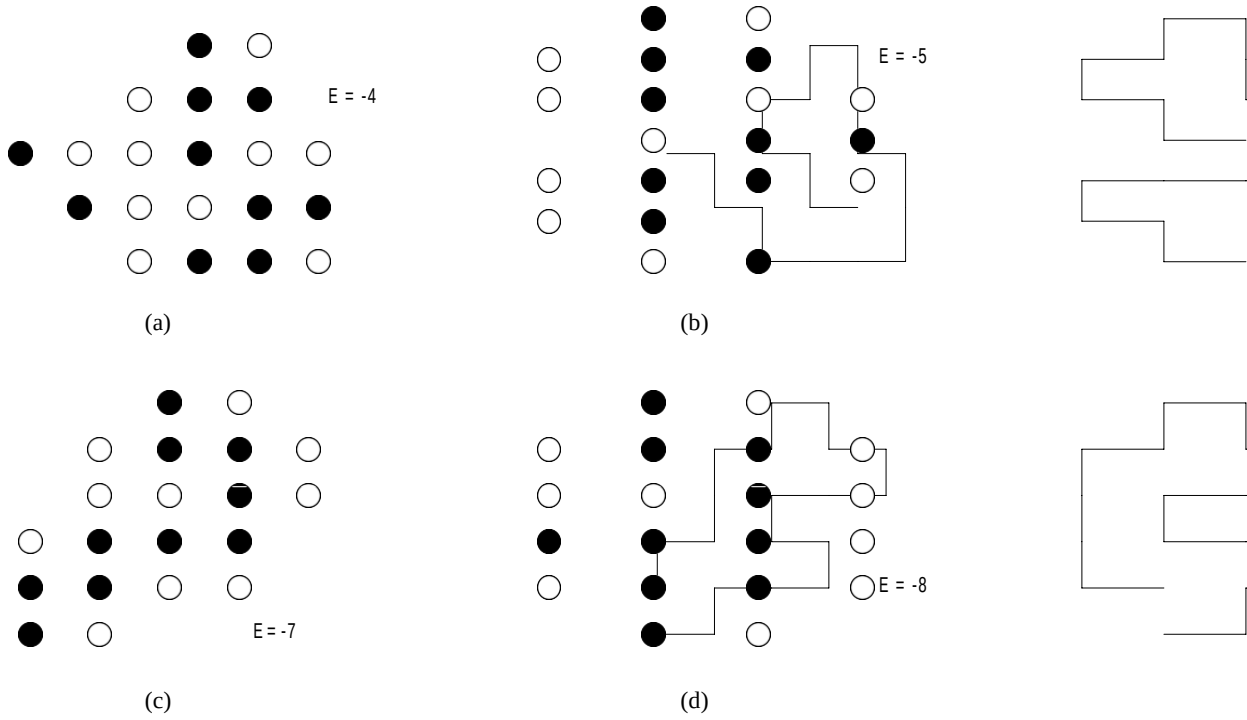

Figura 2.1: Distintas representaciones de una secuencia de tamaño 20, cada una con valor de energía diferente.

La forma más simple de la función de energía para el modelo *HP*, es contar el número de contactos *H-H*, cada contacto topológico *H-H* (*cth*) posee un valor de energía de *1*, mientras tanto cualquier otra interacción (*H-P, P-H, P-P*) tiene una valor de 0. Un *cth* sólo existe cuando dos aminoácidos son topológicamente vecinos pero no se encuentran conectados, es decir, que no son consecutivos en la cadena de aminoácidos, tal y como se muestra en la Figura 2.2. Para este modelo, la conformación nativa es entonces aquella que

maximice el número de contactos *H-H*, y dado que en la naturaleza el estado nativo de una proteína es donde la energía se encuentra en el mínimo absoluto. Podemos definir el valor de la función de energía de una conformación como:

$$E(c) = -\sum_{S_i,S_j \in S} \epsilon(S_i, S_j) \tag{2.1}$$

Donde $\epsilon(S_i, S_j)$ son los contactos topológicos entre un aminoácido hidrofóbico i y un aminoácido hidrofóbico j en un secuencia S.

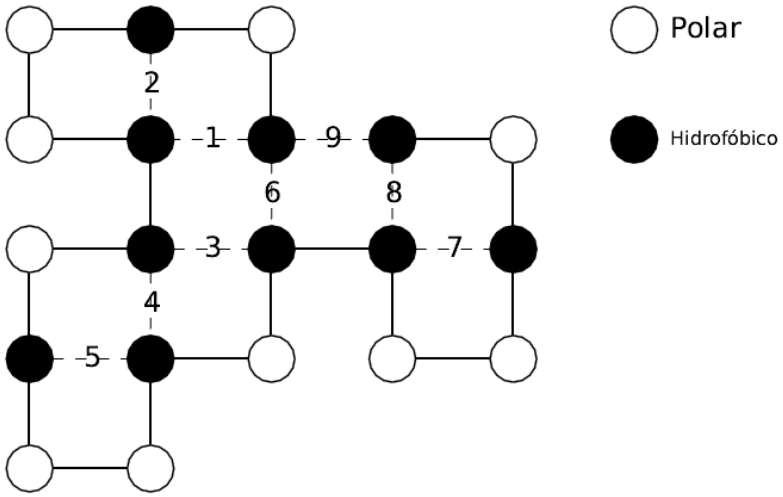

Figura 2.2: Conteo de aminoácido hidrofóbicos para una cadena de tamaño 20.

Originalmente el modelo *HP* fue pensado como un sistema de coordenadas de dos dimensiones, no obstante, el modelo puede ser fácilmente representado en tres dimensiones. En un modelo en 3D, existen más conformaciones por par de enlaces *(z-1 = 5)*, de manera similar cada aminoácido tiene más vecinos espaciales con los que interactuar. No obstante, a pesar de estas diferencias cuantitativas, el comportamiento cualitativo será similar para los modelo en 2D y 3D. Aun así, un modelo en 2D tiene ciertas ventajas sobre el 3D, entre ellas, el obvio hecho de que el costo computacional es menor para el modelo en dos dimensiones.

Las instancias estándar de prueba para el modelo *HP*, Tablas 2.1, 2.2, han sido la base para comprobar la habilidad de las metaheurísticas propuestas en la literatura, aunque las instancias representan un reto para las ciencias computacionales, para muchos biólogos es considerado insatisfactorio la resolución de este problema por medio de este enfoque [13]. Dentro de las siguientes tablas se recopila la información básica

de las instancias estándar, tales como: nombre, Secuencia de aminoácidos *H* y *P*, longitud de la cadena de aminoácidos y el mejor optimo conocido para tal instancia (*E**).

Tabla 2.1: Instancias para el modelo *HP* en 2D

Nombre	Secuencia	L	E*
2d1	$H_2P_5H_2P_3HP_3HP$	18	-4
2d2	$HPHPH_3P_3H_4P_2H_2$	18	-8
2d3	$PHP_2HPH_3PH_2PH_5$	18	-9
2d4	$HPHP_2H_2PHP_2HPH_2P_2HPH$	20	-9
2d5	$H_3P_2HPHPHP_2HPHPHP_2H$	20	-10
2d6	$H_2P_2HP_2HP_2HP_2HP_2HP_2HP_2H_2$	24	-9
2d7	$P_2HP_2H_2P_4H_2P_4H_2P_4H_2$	25	-8
2d8	$P_3H_2P_2H_2P_5H_7P_2H_2P_4H_2P_2HP_2$	36	-14
2d9	$P_2HP_2H_2P_2H_2P_5H_{10}P_6H_2P_2H_2P_2HP_2H_5$	48	-23
2d10	$H_2(PH)_4H_3P\ \ (HP_3)_3(P_3H)_3PH_4(PH)_4H$	50	-21
2d11	$P_2H_3PH_8P_3H_{10}PHP_3H_{12}P_4H_6PH_2PHP$	60	-36
2d12	$H_{12}PHPH(P_2H_2P_2H_2P_2H)_3PHPH_{12}$	64	-42
2d13	$H_4P_4H_{12}P_6(H_{12}P_3)_3HP_2H_2P_2H_2P_2HPH$	85	-53
2d14	$P_6HPH_2P_5H_3PH_5PH_2P_4H_2P_2H_2PH_5PH_{10}P$ $H_2PH_7P_{11}H_7P_2HPH_3P_6HPH_2$	100	-48
2d15	$P_3H_2P_2H_4P_2H_3PH_2PH_2PH_4P_8H_6P_2H_6P_9H$ $sPH_2PH_{11}P_2H_3PH_2PHP_2HPH_3P_6H_{3s}$	100	-50

Longitud: *L*, Mejor óptimo conocido para la instancia: *E*

Tabla 2.2: Instancias para el modelo *HP* en 3D

Nombre	Secuencia	L	E*
3d1	$HPHP_2H_2PHP_2HPH_2P_2HPH$	20	-11
3d2	$H_2P_2HP_2HP_2HP_2HP_2HP_2HP_2H_2$	24	-13
3d3	$P_2HP_2H_2P_4H_2P_4H_2P_4H_2$	25	-9
3d4	$P_3H_2P_2H_2P_5H_7P_2H_2P_4H_2P_2HP_2$	36	-18
3d5	$P_2H_3PH_3P_3HPH_2PH_2P_2HPH_4PHP_2H_5PHPH_2P_2H_2P$	46	-32
3d6	$P_2HP_2H_2P_2H_2P_5H_{10}P_6H_2P_2H_2P_2HP_2H_5$	48	-31
3d7	$H_2(PH)_4H_3P\ \ (HP_3)_3(P_3H)_3PH_4(PH)_4H$	50	-32
3d8	$PH(PH_3)_2P\ \ (PH_2PH)_2H(HP\ )_3(H_2P_2H)_2PHP_4(H(P_2H)_2)_2$	58	-44
3d9	$P_2H_3PH_8P_3H_{10}PHP_3H_{12}P_4H_6PH_2PHP$	60	-52
3d10	$H_{12}PHPH(P_2H_2P_2H_2P_2H)_3PHPH_{12}$	64	-55
3d11	$P\ \ (HPH_2PH_2PHP_2H_3P_3)_3(HPH)_3P_2H_3P$	67	-56
3d12	$P\ \ (HPH)_3P_2H_2(P_2H)_6H(P_2H_3)_4P_2(HPH)_3P_2HP\ (PHP_2H_2P_2HP\ )_2$	88	-72

2.1.1. Funciones de energía multi-objetivo

Desde que el modelo *HP* fue creado por Dill [17], varias funciones de energía han surgido como alternativa, con la finalidad de lograr una mejor discriminación entre las estructuras proteicas que pueden existir dentro de los límites del modelo. Cada una de estas funciones plantea una forma diferente de atacar el problema. Dentro de estas funciones, tenemos las funciones presentadas en [25, 27, 24]. Las cuales son funciones multi-objetivo, creadas a partir de una función mono-objetivo, utilizando el método de descomposición. Asegurando siempre, que la suma de sus objetivos debe ser igual al valor de la función mono-objetivo original. A continuación se presenta una breve descripción de cada una:

- *Función Parity*: Esta función de energía enumera la cadena de aminoácidos y ve estos como número pares y nones, por lo que el primer objetivo de esta función cuenta aquellas iteraciones hidrofóbicas presentadas por los aminoácidos *i* pares, y el segundo objetivo toma aquellas donde intervienen los aminoácidos *i* impares

$$f_1(c) = -\sum_{S_i,S_j \in S | i<j} e_p(S_i, S_j) \; Donde \; i \equiv 0 \mod 2 \tag{2.2}$$

$$f_1(c) = -\sum_{S_i,S_j \in S | i<j} e_p(S_i, S_j) \; Donde \; i \equiv 1 \mod 2 \tag{2.3}$$

- *Función Locality*: Esta función al igual que la función de Parity, enumera la cadena de aminoácidos, para evaluar la estructura se requiere de un factor de localidad δ, con base en esto el primer objetivo cuenta todas las iteraciones hidrofóbicas presentadas entre dos aminoácidos cuya resta del valor de sus posiciones sea menor al factor de localidad, y el segundo se cumple cuando dos aminoácidos su diferencia es mayor al factor.

$$f_1(c) = -\sum_{S_i,S_j \in S | i<j} e_l(S_i, S_j) \; Donde \; j - i \leq \delta \tag{2.4}$$

$$f_1(c) = -\sum_{S_i,S_j \in S | i<j} e_l(S_i, S_j) \; Donde \; j - i > \delta \tag{2.5}$$

- *Función H-subset*: Esta función divide los aminoácidos hidrofóbicos en dos grupos, y evalúa las iteraciones presentas por los miembros del mismo grupo como el primer objetivo, y las iteraciones entre grupos como segundo objetivo.

$$f_1(c) = -\sum_{S_i,S_j \in H_1} e_h(S_i, S_j) + -\sum_{S_i,S_j \in H_2} e_h(S_i, S_j) \tag{2.6}$$

$$f_1(c) = -\sum_{S_i \in H_1,S_j \in H_2} e_h(S_i, S_j) \tag{2.7}$$

2.2. Optimización.

La optimización es el proceso de buscar la mejor manera de realizada una actividad. Matemáticamente hablando, la optimización es el proceso de maximizar o minimizar una función mientras satisface determinadas restricciones. Un problema de optimización se conforma por una o varias funciones objetivo y una, ninguna o varias restricciones [11, 10]. Este tipo de problemas se describen mediante los siguientes elementos:

Variables de decisión: Son cantidades numéricas cuyos valores son elegidos en un problema de optimización, las *n* cantidades seccionadas se denotan como $x_j(j = 1, 2, \ldots n)$. El vector de diseño *X* de *n* variables se representa por:

$$X = x_1, x_2, \ldots, x_n \tag{2.8}$$

Restricciones. En la mayoría de los problemas de optimización, las características particulares del entorno, imponen ciertas restricciones. Para que una solución sea considerada aceptable debe cumplir con *m* restricciones y *p* igualdades. Las restricciones describen dependencias entre las variables de decisión y los parámetros específicos del problema. Estas restricciones son expresadas en forma de desigualdades matemáticas:

$$g_i(X) \leq 0, i = \{1, \ldots, m.\} \tag{2.9}$$

O en la forma de igualdades como:

$$h_j(X) = 0, j = \{1, \ldots, p.\} \tag{2.10}$$

En la ecuación 2.10 aclaramos, $p \leq n$ ya que de otro modo el problema no tendría grados de libertad y estaría sobreentrenado, o en otras palabras habrá más incógnitas que ecuaciones a resolver. Estas restricciones pueden presentarse de forma explícita, es decir dispondremos de una fórmula para realizar el cálculo o puede presentarse de forma implícita, en cuyo caso se contará con un algoritmo que nos facilite dichos cálculos para $g_i(X)$ para un vector *X*.

Función objetivo. Las funciones objetivo se expresan como: $f_1(x), f_2(x), \ldots, f_k(x)$ donde k es el número de funciones objetivo que se desea resolver dentro del problema. Claramente, el conjunto de funciones objetivo forman un vector k-dimensional:

$$F_{(X)} = [f_1(X), f_2(X), \ldots, f_k(X)]^T \tag{2.11}$$

Cuando un problema involucra un único objetivo, la tarea de encontrar soluciones óptimas se denomina optimización mono-objetivo. Mientras que, cuando el problema abarca más de un objetivo, la tarea de encontrar una o más soluciones óptimas se denomina optimización multi-objetivo.

2.3. Optimización Multi-objetivo.

Como el nombre sugiere, un problema de optimización multi-objetivo (*MOP*), trabaja con más de una función objetivo, en la mayoría de los problemas, los múltiples criterios son fácilmente reconocibles. En los problemas de optimización mono-objetivo, la meta es buscar una solución que optimice esa función, Siguiendo por ese camino podemos asumir erróneamente, que la optimización multi-objetivo es buscar una solución para cada objetivo. Informalmente pude ser descrito como el problema de buscar: "Un vector de variables de decisión perteneciente a algún universo (Ω), que satisfaga las restricciones ya sean m desigualdades ($g(X)$) o p igualdades ($h(X)$) que optimice una función vectorial cuyos elementos representan a las funciones objetivo". Estas funciones forman una descripción matemática de distintos criterios que, usualmente, estarán en conflicto unos con otros. Por lo tanto, el término optimizar significa encontrar una solución tal que obtenga valores aceptables para el tomador de decisiones para todas las funciones objetivos [10, 15]. Formalmente podemos definirlo de la forma siguiente:

Un problema de optimización multi-objetivo general se define como la tarea de minimizar (o maximizar) k funciones objetivo:

$$F_{(X)} = [f_1(X), f_2(X), \ldots, f_k(X)]^T \tag{2.12}$$

Sujeto a:

$$\begin{aligned} g_i(x) &\leq 0, i = \{1, \ldots, m\} \\ h_j(x) &= 0, j = \{1, \ldots, p\} \end{aligned} \tag{2.13}$$

Donde:

$$X \in \Omega \tag{2.14}$$

precisamente un problema multi-objetivo son esos problemas donde el objetivo es optimizar *k* funciones objetivo de forma simultánea. La forma de optimizar se define como encontrar una solución que minimiza (o maximiza) los componentes del vector *F(x)*, donde *x* es un vector de diseño de *n* dimensiones, con *n* variables de decisión, $x = (x_1, x_2, ..., x_n)$ de algún Ω. Observemos que $g_i(x) \leq 0$ y $h_j(x) = 0$ representan restricciones que deben ser resueltas en forma simultánea mientras se minimiza (o maximiza) *F(x)* y Ω contiene todos los posibles *x* que pueden ser usados para satisfacer *F(x)* [11].

2.4. Óptimalidad de Pareto.

La existencia de múltiples funciones objetivo, causa un conflicto de intereses, haciendo que el concepto de óptimo cambie. Debido a que la finalidad de los problemas de optimización multi-objetivo es encontrar un conjunto de soluciones que representen un compromiso entre los objetivos, en lugar de una única solución. Surge el concepto de óptimalidad comúnmente llamada óptimalidad de *Edgeworth-Pareto,* también conocida simplemente como óptimalidad de Pareto [10, 11]. A continuación se presentan las definiciones para comprender la óptimalidad de Pareto, en las siguientes definiciones tomamos que $x, y \in X$ denotan dos vectores de decisión y, se asume un problema de minimización.

Definicion **2.1** *Óptimo de Pareto: Un vector de diseño 'x' es óptimo de Pareto si no existe otro vector de diseño 'y' tal que:*

$$f_i(y) \leq f_i(x), \forall i \in \{1, \ldots, k\} \; y \; f_j(y) < f_j(x), \exists j \in \{1, \ldots, k\} \tag{2.15}$$

En palabras, *'x'* es un óptimo de Pareto si no existe otro vector factible *'y'* que sea mejor o a lo mucho igual para cada función y estrictamente que al menos una función de ellos sea peor.

Definicion **2.2** *Conjunto de óptimos de Pareto: El conjunto de óptimos de Pareto, denotado por P*, se define como:*

$$P* = \{x \in X \mid x \text{ es óptimo de Pareto}\} \tag{2.16}$$

Se dice que todos los vectores que cumplan con la definición 2.1 pertenecen al conjunto de las mejores soluciones para un problema de optimización multi-objetivo.

Para comprender la siguiente definición, primero debemos comprender que los MOPs, trabajan con 2 espacios euclidianos:

- El espacio *n-dimensional* conocido como espacio de decisión (espacio *X*), conformado por las variables de decisión en la cual cada coordenada corresponde a un componente del vector *X*.
- El espacio *k-dimensional* de las funciones objetivo (espacio *Z*) en el cual cada coordenada corresponde al componente del vector $f_k(X)$

La función objetivo transforma los vectores de decisión desde el espacio de búsqueda hacia un espacio objetivo [41] Figura 2.3. De esta forma, dado un problema de minimización, el objetivo es encontrar aquella solución que sea óptimo de Pareto.

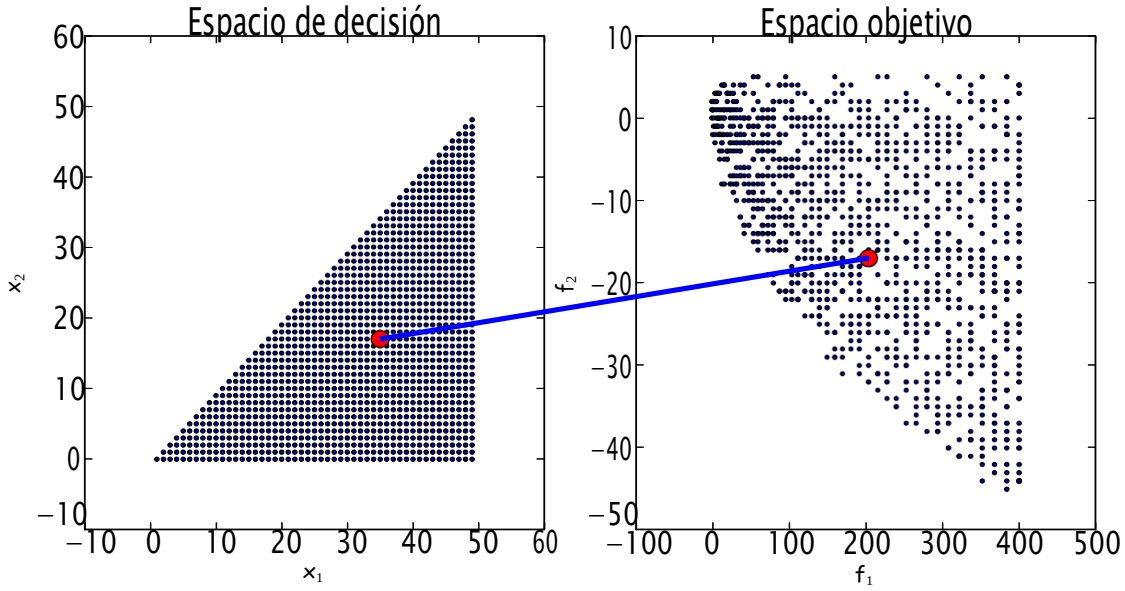

Figura 2.3: Mapeo de una solución $x \in X$, cuya imagen $z = f(x)$.

Definicion **2.3** *Frente óptimo de Pareto: El frente óptimo de Pareto, denotado por PF*, corresponde a la imagen del conjunto óptimo de Pareto en el espacio objetivo y se define como:*

$$FP* = \{\boldsymbol{f}(x) \in Z \mid x \in P*\} \tag{2.17}$$

Las soluciones que son óptimos de Pareto son también conocidas como soluciones admisibles, o eficientes; cuyos correspondientes vectores se les denomina *no dominados*. Son vectores que aparentemente no tiene ninguna relación clara además de su pertenencia a *P**, el cual es el conjunto de todas las soluciones cuyos vectores correspondientes después de ser evaluados por la función objetivo son no dominados con respecto a todos los otros vectores [51]. Cuando se traza en el espacio objetivo, el conjunto óptimo de Pareto

*P**, se conoce como el frente óptimo de Pareto *FP**, las cuales son las mejores soluciones conocidas para el problema.

Resumiendo *P** es el subconjunto de los vectores de decisión que al ser evaluados, se representan en el espacio objetivo como el conjunto de las mejores soluciones conocidas para un problema optimización también llamado como Frente óptimo de Pareto (*FP**). Por lo general no es fácil encontrar una expresión analítica de la superficie que contiene *PF**, por lo que evaluar la mayoría puntos posibles es la única forma de reconocer el frente de Pareto [10, 11]. Este proceso queda ilustrado en la Figura 2.4, donde se tiene un universo de vectores de decisión, donde son evaluados mediante un vector de funciones objetivos (las cuales se requiere minimizar), quedando representados como puntos en el espacio objetivo, obteniendo así un frente de Pareto denotado por la linea roja.

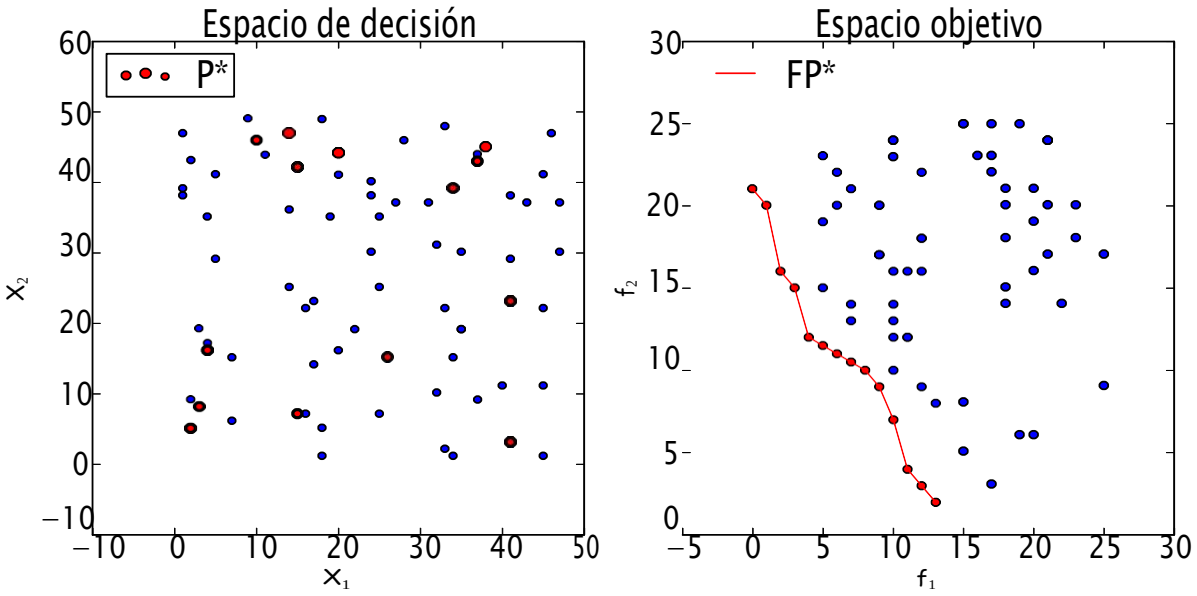

Figura 2.4: Frente de Pareto, para un problema de optimización multi-objetivo con dos funciones objetivos que se requieren minimizar.

Para conocer cuáles son las soluciones que conforman el frente óptimo de Pareto, debemos comprender el concepto de *Dominancia*, el cual se utiliza para comparar dos posibles soluciones $x, y \in X$ y concluir cuál de las candidatas es mejor que la otra, en otras palabras cuál de las dos satisface mejor los conflictos entre los objetivos. A continuación se muestran las situaciones en las que las dos soluciones pueden quedar al momento de ser confrontadas.

Definicion **2.4** *Dominancia débil de Pareto. Se dice que una solución* x *domina débilmente a una solución* y, *denotado por:*

$$x \preceq y, \ si \ y \ sólo \ si \ f_i(y), \forall \ i \ \in \{1, \ldots, k\} \tag{2.18}$$

Definicion **2.5** *Dominancia de Pareto. Se dice que una solución x domina a una solución y.*

$$x \prec y, \ si \ y \ sólo \ si \ f_i(x) \leq f_i(y) \wedge f_j(x) < f_j(y), \forall i \in \{1, \ldots, k\}, \exists j \in \{1, \ldots, k\} \tag{2.19}$$

Definicion **2.6** *Dominancia estricta de Pareto. Se dice que una solución x domina estrictamente a una solución y, denotado por:*

$$x \prec_s y, \ si \ y \ sólo \ si \ f_i(x) < fi(y) \forall i \in \{1, \ldots, k\} \tag{2.20}$$

Definicion **2.7** *Incomparabilidad. Dos soluciones* x *y* y *son incomparables si* 'x' *no domina a* 'y' *de ninguna forma y ,* 'y' *no domina a* 'x' *de ninguna forma, denotado por* x || y.

$$si \ y \ sólo \ si \ \mathrm{x} \not\prec \mathrm{y} \wedge \mathrm{y} \not\prec \mathrm{x}. \tag{2.21}$$

Definicion **2.8** *Indiferencia. Dos soluciones* 'x' *y* 'y' *son indiferentes*

$$si \ y \ sólo \ si \ f_i(x) = fi(y), \forall i \in \{1, \ldots, k\}.$$

Como ya se ha mencionado anteriormente, resolver un problema de optimización multi-objetivo consiste en encontrar un conjunto de soluciones que satisfaga los requisitos de las funciones objetivos, el cual puede contener una infinidad de alternativas. Sin embargo, se tendrá que elegir una de esas soluciones como resultado a nuestro problema. Matemáticamente hablando, todas las soluciones en el óptimo de Pareto son igualmente aceptables, por lo que elegir una de ellas requiere más del conocimiento que nos puedan proporcionar las funciones objetivos planteadas para el problema. Por lo tanto se requiere una persona (o conjunto de personas) que tome(n) las decisiones, pues se asume que tiene(n) un amplio conocimiento del problema y podrá(n) hacer una diferencia entre las diferentes soluciones para tomar una decisión final [20].

2.5. Computación Evolutiva.

La computación evolutiva (*Evolutionary Computing, EC*) es el área de investigación dentro de las ciencias computacionales conformada por un conjunto de algoritmos bioinspirados capaces de abordar problemas

de optimización. Que como su nombre sugiere, está inspirado en el proceso natural de la evolución biológica, en donde una población de individuos es ubicada en un nicho, la cual se esfuerza para adaptarse y sobrevivir en su ambiente [18]. El grado de *aptitud* que cada individuo posee, está determinado por el habitad y se define como que tan bien un individuo ha completado sus metas, las cuales representan las probabilidades de sobrevivir y reproducirse. En el contexto de la *EC*, cada individuo representa una solución potencial, en donde su calidad está en función de su aptitud, la calidad determina la posibilidad de que una solución se mantenga y sea usada además como semilla para concebir futuras soluciones candidatas. La relación entre los conceptos básicos del cómputo evolutivo y la resolución de un problema de optimización se muestran en la Tabla 2.3.

Tabla 2.3: La básica relación entre la metáfora del cómputo evolutivo contra la resolución de un problema

Evolución		Resolución
Entorno	$\longleftrightarrow$	Problema
Individuo	$\longleftrightarrow$	Solución potencial
Aptitud	$\longleftrightarrow$	Calidad

La idea de aplicar los principios del *Darwinismo* para resolver problemas de forma autónoma, data desde la década de los cuarenta. En 1948, *Turing* propuso *una búsqueda genética o evolutiva* [18]. A lo largo de los años varias propuestas fueron surgiendo alrededor de la misma idea básica. Durante la década de los sesenta, tres diferentes implementaciones fueron desarrolladas en diferentes lugares. En EE. UU. Fogel et al. Introdujeron la *programación evolutiva* [22], mientras tanto *Hollan* llamo a su método *algoritmo genético* [33], por otro lado, en Alemania Rechenberg et al. presentaron *estrategias evolutivas* [19]. Durante la década de los noventa Koza presentó una cuarta idea, la cual se conoce como *programación genética* [35]; durante años todos estos conceptos fueron trabajados de forma independiente, hasta que a principios de los noventa, dichas ideas fueron vistas como diferentes representaciones de una misma tecnológica conocida como *computación evolutiva*, Así como cualquier algoritmo diseñada bajo estos principios se les denoto algoritmos evolutivos (*evolutionary algorithm*, EA). El objetivo de los EAs es realizar tareas de optimización y aprendizaje con la capacidad evolucionar [55]. Los cuales tienen tres características principales:

- **Basado en Población**. Los EAs mantienen un grupo de soluciones, llamados *poblaciones*, para optimizar el problema de manera paralela. Las poblaciones es el principio más básico del proceso evolutivo.

- **Orientado a la aptitud**. Cada solución en una población es llamado *individuo*, cada individuo está formado por su representación en forma de genes y su valor de aptitud. Usualmente los EAs prefieren los individuos más aptos, puesto que son la base de la optimización y la convergencia de los algoritmos.

- **Variación impulsada**. Los individuos serán sometidos a una serie de operaciones de variación para simular los cambios genéticos de genes, lo cual es fundamental para buscar en el espacio de soluciones.

Hoy en día cada vez es más difícil distinguir las diferencias entre los distintos tipos de algoritmos evolutivos existentes. Cada una se originó de manera independiente y con diversas motivaciones inspiradas en diferentes ambientes [9]. En la siguiente sección se describirá el funcionamiento y los componentes de un algoritmo genético, puesto que para propósito de la investigación es el que más nos interesa. Sin embargo, cabe mencionar que los mismos principios aplican para otros algoritmos.

2.5.1. Algoritmos genéticos.

Un algoritmo genético (*Genetic Algorithm*, GA) es un método para resolver problemas teniendo como modelo los principios de la genética y la evolución, basándose primordialmente en la ideal de la supervivencia del más apto. Es una técnica útil para los distintos problemas de búsqueda y optimización [48]. La ciencia que trata de los mecanismos responsables de similitudes y diferencias en una especie se llama genética, la genética nos ayuda a diferenciar entre la herencia y variaciones que existe entre los pobladores de una misma especie. Los conceptos de los algoritmos genéticos se derivan directamente de la evolución natural, por lo que comprender los principales conceptos que participan tras el fondo biológico de las especies ayuda a comprender el funcionamiento de los algoritmos genéticos, los cuales se detallan a continuación:

- **Cromosomas.** Toda la información genética que un ser vivo pueda tener se encuentra almacenada en los cromosomas, cada cromosoma se encuentra dividido en pequeñas partes llamadas *genes*. Los genes codifican las características que un individuo pueda tener. Los posibles valores que un gen pueda tener se les denominan *alelos* [48]. El conjunto de todos los posibles alelos que un gen pueda tener, determina todas las posibles variaciones que las futuras generaciones puedan tener. La cantidad de genes en un cromosoma determinan la diversidad de los individuos en la poblacional. En la naturaleza los genes se encuentran repartidos en diferentes cromosomas, mientras tanto en los GA, todos los genes usualmente se encuentran en un único cromosoma

- **Genética.** Un punto importante para comprender la genética, tiene con ver la capacidad de distinguir entre el genotipo y el fenotipo. Para un único individuo, la entera combinación de sus genes se conoce

como genotipo. El fenotipo por otra parte es el aspecto físico que se da al decodificar el genotipo. En el ámbito de la evolución la selección siempre se da con base en el fenotipo, mientras que la reproducción recombina el genotipo. En un GA las soluciones candidatas se codifican de acuerdo a su genotipo y se evalúan de acuerdo a su fenotipo, lo que es lo mismo el genotipo es el cromosoma en su estado puro sin interpretar, mientras tanto el fenotipo es la comprensión del genotipo como una solución potencial al problema [45].

- **Reproducción.** La reproducción de las especies, puede darse por diferentes medios, la reproducción puede darse de forma asexual, que es lo que ocurre cuando una célula se divide y la misma información genética es copiada a la siguiente generación, en los GA a este proceso se le conoce como elitismo y de forma sexual cuando dos individuos se juntan para recombinar sus genes y formar un nuevo individuo con las mejores partes de ambos.

- **Selección natural.** El origen de las especies está basada en la idea de la preservación de las variaciones favorables y la exclusión de las variaciones menos favorables. La variación se refiere a las diferentes caras que se puede presentar entre los individuos de una misma especie [48]. Mientras los individuos más favorables son seleccionados para dejar descendencia, las nuevas generaciones serán más aptos para sobrevivir, por lo tanto conforme avance el tiempo la lucha por la supervivencia la gana aquellos individuos que nacieron con ciertas ventajas. Una abstracción en los GA, se da cuando se descartan las soluciones menos adecuadas para el problema y sólo conservamos las mejores.

En un problema de optimización al conjunto de todas las posibles soluciones se le conoce como espacio de búsqueda. Básicamente el problema consiste en encontrar la solución que se adapte mejor que cualquier otra en el espacio. Si se pudiera enumerar rápidamente todas las posibles soluciones el problema no representaría ninguna dificultad, por otro lado cuando el espacio de búsqueda se hace grande, hacer dicha enumeración sería imposible por la cantidad de tiempo que tomaría [29]. Para este caso es cuando un algoritmo genético es útil para encontrar la solución óptima.

Los GA mantienen un conjunto de posibles soluciones, cada una representada en forma de cromosoma, además los algoritmos genéticos requieren un conjunto de operadores de reproducción que operen directamente sobre los cromosomas, dentro de estos operados los GA enfatizan la importancia de la cruza sexual (operador principal) sobre la mutación (operador secundario) [9]. La apropiada implementación de estos algoritmos determinará el comportamiento del GA puesto que depende extremadamente de esto, usualmente es difícil encontrar un representación adecuada que sea coherente y relevante acorde a las necesidades del problema.

Manteniendo un conjunto de soluciones candidatas, se debe precisar un método que la distinga y confirme

cuál de ellas es la solución óptima. El método de selección es capaz de comparar cada individuo en la población con base en su función de aptitud; durante la fase de evaluación a cada cromosoma se le asigna un valor de aptitud que corresponde que tan buena es la solución candidata, La solución óptima para el problema es aquella que minimice o maximice la función objetivo. A continuación, el Algoritmo 1 muestra los pasos esenciales para desarrollar un algoritmo genético.

Algoritmo 1 Algoritmo Genético Simple

1: *top*:
2: $P \leftarrow INITIALIZE(P)$
3: *loop*:
4: **while** *Condición de paro* **do**
5: $Q \leftarrow PARENTSELECTION(P)$
6: $Q \leftarrow GENETICOPERATIONS(Q)$
7: $Q \leftarrow EVALUATION(Q)$
8: $P \leftarrow GENERATIONUPDATE(Q)$
9: ***end while***

Con la reproducción y la función de aptitud claramente establecidas, un algoritmo genético evoluciona de acuerdo al estructura básica, mostrada en el Algoritmo 1. El proceso empieza generando de forma aleatoria una población inicial de cromosomas (P), con un buen tamaño para que exista una buena diversidad en el material genético. Luego el proceso continúa dentro de un ciclo, con la intención de evolucionar la población. Cada iteración consiste de los siguientes pasos:

- **Selección**. La primera parte consiste en seleccionar los individuos que pasaran a la reproducción, seccionando aquellos usualmente con el mejor valor de aptitud.
- **Reproducción**. En el segundo paso, se genera la población descendiente (Q), resultado de los métodos de variación de cruza y mutación.
- **Evaluación**. Los nuevos individuos son evaluados.
- **Reemplazo**. En el último paso, los individuos padres son eliminados y la descendencia se convierte en los nuevos padres ($P \leftarrow Q$).

El algoritmo se detiene cuando la población converge hacia la solución óptima, o una solución que haya caído en un mínimo local. En la fase de reproducción el material genético de dos o más padres es combinado

para obtener uno o más herederos. La mutación ayuda a mejorar un individuo produciendo una nueva versión de él [48]. En resumen los algoritmos genéticos son sistemas basados en el supuesto funcionamiento de la vida, manteniendo una población de individuos, hacer que evoluciones y quedarse con los más aptos.

Los GA no garantizan encontrar la solución óptima, dado que es un proceso estocástico y es posible que la selección de genes se encuentre muy lejos de lo querido. Sin embargo estos algoritmos son extremadamente eficientes y son usados en diferentes campos de estudio.

2.5.2. Algoritmos evolutivos para resolver problemas multi-objetivo.

El objetivo de los algoritmos genéticos tradicionales es encontrar una única solución para los problemas de optimización, la cual maximice o minimice un valor de aptitud directamente relacionado a una medida de calidad. Existen problemas donde la calidad de la solución se encuentra definida por la relación, posiblemente en conflicto de diversos objetivos. Para esta clase de problemas, los algoritmos genéticos tuvieron que ser adaptados para ser competentes para encontrar soluciones que cumplieran con los requisitos de los objetivos, a este tipo de algoritmos se les conoce como *algoritmo evolutivo multi-objetivo (Multi-Objective Evolutionary Algorithm, MOEA)* [11].

Existen una gran variedad de MOEAs, pero la mayoría de ellos utilizan el concepto de *Pareto* para identificar con base en la aptitud de las soluciones, aquellas que son no dominadas, las cuales conforman el conjunto de parteo. Para que un MOEA pueda cumplir con las expectativas para resolver problemas multi-objetivo debe de cumplir con los siguientes requisitos:

- Preservar las soluciones no dominadas, con $PF_{actual} \rightarrow PF_{conocido}$.
- Guiar el $PF_{conocido}$ hacia el mejor frente.
- Generar y mantener la diversidad en los puntos del frente de Pareto.
- Proveer al tomador de decisiones con un limitado número de puntos en frente de Pareto

Con esto un MOEA debe ser capaz de alcanzar el Frente óptimo de Pareto, manteniendo una buena diversidad en las soluciones y previniendo la pérdida de buenas soluciones. En el siguiente capítulo se muestra el desarrollo del MOEA, utilizado en este trabajo el cual está basado en el trabajo de K. Deb el algoritmo *NSGA-II* [14].

3

Metodología

3.1. NSGA-II.

El algoritmo *Non dominated Sorting Genetic Algorithm II* (NSGA-II) [14], es una versión mejorada de su predecesor, el NSGA, reduciendo la complejidad computacional. Se clasifica como un algoritmo altamente elitista, ya que incorpora un método que preserva las soluciones claves a lo largo de las generaciones que el algoritmo pueda tener. El proceso comienza con una población inicial de los padres P_t, construida al azar y de tamaño N, la población descendiente Q_t (tamaño N) se genera en primera instancia usando la población de padres y está determinado por los mecanismos de selección, cruce y mutación definidos por el algoritmo genético clásico. Ya obtenida la población Q_t, ambas poblaciones son combinadas para generar la población R_t de tamaño $2N$. Luego, R_t se somete a un método de ordenamiento basado en no dominancia, el cual clasifica la población R_t en diferentes frentes de Pareto $\{F_1, ..., F_k\}$. Una vez que el proceso de ordenamiento no dominado ha terminado, por cada frente se calcula un valor de distancia (*crowding-distance*) para cada individuo dentro de los frentes. La nueva generación de padres se genera con base en los individuos pertenecientes a los frentes no dominados. La nueva población P_{t+1} comienza a ser construida con el mejor frente no dominado F_1, continuando con los individuos del segundo frente F_2, el tercero F_3 y así sucesivamente. Debido a que la población R_t es de tamaño $2N$ y la nueva población de padres P_{t+1} requiere solamente N soluciones, no todas las soluciones en los frentes pertenecientes a la población R_t pueden ser alojados en la población P_{t+1}, dichos frentes que no puedan ingresar en P_{t+1} serán eliminados de la ejecución del algoritmo. Al momento de generar la población P_{t+1}, puede darse el caso de que el último frente F_l, contenga m soluciones más, que el necesario para completar el tamaño la población, si es el caso, son eliminados aquellos individuos con el menor valor de *crowding-distance*, quedándose únicamente con la cantidad de individuos requeridos.

Para mantener una población diversa, el algoritmo NSGA-II cuenta con una métrica para estimar la densidad de la población llamada *crowding-distance*. Además dentro de la implementación desarrollada para esta investigación se agregó una función de densidad auxiliar basada en el método *fingerprint*, propuesto por Chira et al. [8], el cual a diferencia de la *crowding-distance* se basa en los contactos topológicos entre los aminoácidos indistintos a su clase; aun así este método funciona más como mecanismo de selección de sobrevivientes, pues si el valor de *fingerprint* es bajo, el individuo es remplazado con uno aleatorio.

Hablando de procedimientos para seleccionar sobrevivientes, además del acotamiento de la población R_t, se trabajó con la idea de proporcionar una esperanza de vida a cada individuo, el cual tras cumplirse sus n generaciones de vida, el individuo se retira de la población y se sustituye por uno generado al azar. Lo cual no afecta al individuo posicionado como la mejor solución dentro del primer frente [18].

Para este trabajo, los mecanismos de cruza y mutación están enriquecidos con la filosofía *hill-climbing* [43, 8], además, utiliza dos tipos de mutación, con el fin de perturbar de manera más ordenada las soluciones, donde las mejores soluciones utilizan la mutación *Pull-Moves* [38], mientras tanto las soluciones menos adecuadas trabajan con la mutación *multigen* [23]. Aparte con el fin de explorar más el espacio de soluciones, se añadió una Algoritmo *VNS* [1, 40, 30] el cual sólo trabaja con las n mejores soluciones, sin tomar en cuenta la solución ubicada en el primer puesto de la población.

A continuación se desglosan las principales partes del algoritmo, desde cómo se representa las proteínas en forma de cromosomas hasta los mecanismos para mantener la diversidad.

3.1.1. Representación del cromosoma

Para esta versión del algoritmo NSGA-II, cada individuo representa una posible configuración proteica, dada una secuencia específica de aminoácidos. Gracias al modelo *HP* lo único que nos interesa saber acerca de un aminoácido es su respectiva posicion en la cadena proteica para así poder generar un cromosoma de movimientos.

Cada cromosoma es codificado como un arreglo de tamaño *N-1*, donde *N* es el número de aminoácidos en la estructura primaria de la proteína. Dentro del cromosoma cada gen representa la posición X_i del aminoácido o residuo i respecto a su predecesor, i.e. En el gen N_1 se tiene la posición del aminoácido 2 respecto al primer aminoácido. Utilizando un codificación de movimientos absolutos donde los aminoácidos son relativos a los ejes del plano donde se modelan, los genes pueden escoger entre 4 posibles valores para modelos en 2D {R,L,U,D} y 6 posibles direcciones para modelos en 3D {R,L,U,D,F,B}. Estos índices representan las direcciones que un aminoácido puede tomar: Derecha (*R*), Izquierda (*L*), Arriba (*U*), Abajo (*D*), Adelante (*F*) y Atrás (*B*). Además, cada cromosoma, trabaja con un conjunto de coordenadas internas

que representan la posición del amino en la cuadrilla.

En la Figura 3.1 se muestra la representación del cromosoma [U, L, L, D, R, D, L, D, D, R, U, R, U, R, D, R, U, U, L] de longitud 19, en un plano de 2D, para una proteína con una longitud de 20 aminoácidos. Como se observa los movimientos del arreglo empiezan del segundo aminoácido, puesto que el primero se usa de punto de partida para empezar a construir el arreglo de movimientos.

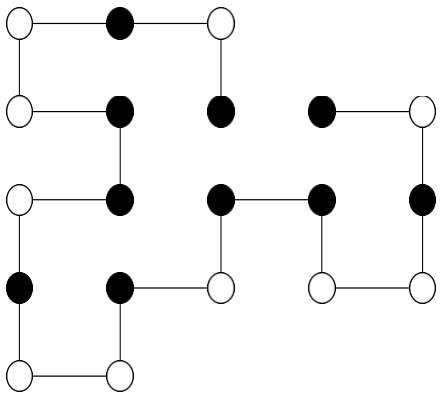

Figura 3.1: Posible conformación que una proteína puede adoptar, correspondiente al cromosoma (U, L, L, D, R, D, L, D, D, R, U, R, U, R, D, R, U, U, L)

3.1.2. Operador de cruzamiento

Durante el proceso de fecundación, el espermatozoide y el óvulo se unen para formar un nuevo organismo, la mitad de estos cromosomas procede de cada padre. Este mismo hecho simulan los algoritmos genéticos con el proceso de cruzamiento donde dos cromosomas comparten información genética con el fin de crear un nuevo individuo. En esencia la idea principal del operador, es combinar las mejores virtudes de 2 soluciones padres, con la esperanza de que produzcan un hijo con mejores aptitudes que las de ellos.

Siguiendo con la idea de combinar los mejores atributos, para este trabajo se utilizó el método de *Path Relinking* [28, 44], desarrollado originalmente como un mecanismo complementario para el algoritmo *búsqueda tabú*. El enfoque de este método es generar nuevas soluciones mediante la exploración de trayectorias entre 2 soluciones candidatas, escogiendo a una de ellas como la *solución inicial*, generar una ruta en el espacio de vecinos hasta llegar a la segunda solución, denominada *solución guía*. La generación de la ruta de soluciones intermedias se da introduciendo los atributos de la solución guía hacia la solución inicial.

En la Figura 3.2 vemos un ejemplo del funcionamiento del *Path Relinking*, donde la solución inicial (*I*) va

generando nuevas soluciones, a partir de los movimientos pertenecientes a la solución guiá (*G*). Suponiendo un caso de minimización el punto sombreado representa la ocurrencia de una solución hija, mejor respecto a sus padres. Como es de costumbre en los algoritmos genéticos, el operador de cruza genera 2 hijos por cada 2 padres. Siguiendo con esa idea, el *Path Relinking* utilizado para este trabajo toma a uno de los padres como solución inicial (*I*) y el otro como guiá (*G*) para generar un conjunto de soluciones de las cuales la mejor de ellas se elige como el primer hijo, para el segundo hijos los padres intercambian su rol como solución inicial y guiá. El Algoritmo 2 resume el funcionamiento del *Path Relinking*.

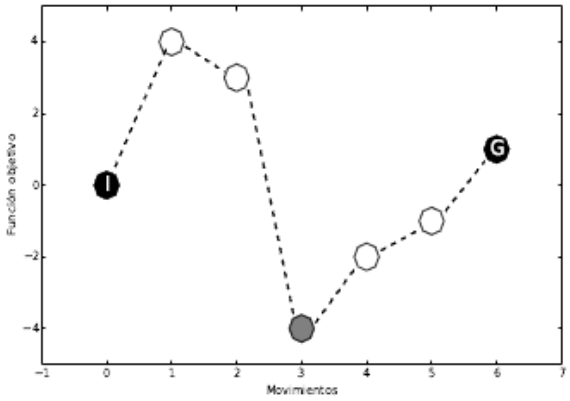

Figura 3.2: Ejemplo de *Path Relinking* donde la línea punteada es un camino generado para unir las dos soluciones. El nodo sombreado es una mejor solución que *I* y *G*.

Algoritmo 2 Cruzamiento

Require: S_i,S_g

1: **procedure** *Path Relinking*
2: S^R-*Ruta de soluciones*
3: $S_{temp} \leftarrow S_i$
4: $indices \leftarrow len(S_i)$
5: **for** $x \in indices$ **do**
6: **if** $S_g[x]! = S_{temp}[x]$ **then**
7: $S_{temp}[x] \leftarrow S_g[x]$
8: $S^R.append(S_{temp}.copy())$
9: $Fitnesses \leftarrow F_{S^R}$-*Evaluar los individuos en* S^R
10: *Return best ind* $\in S^R$

3.1.3. Mutación Multigen

La mutación es una parte fundamental en la evolución. sin ella las especies no evolucionarían. La evolución tiene lugar cuando una nueva versión de un gen, que originalmente surge por una mutación, aumenta su frecuencia y se extiende a la especie gracias a la selección natural [53]. Dentro de los algoritmos genéticos la mutación porta el papel de asegurar que ningún punto en el espacio tenga una probabilidad de cero en ser examinado. Es quien permiten crear nuevos individuos, escapar de los mininos locales, abastecernos de nuevo material genético, y en ciertos casos mejorar los procesos de elitismo en busca de mejores soluciones. Debido al gran espacio de búsqueda que el problema posee, para este trabajo se utilizaron 2 tipos de mutaciones. Una de ellas, es la mutación multigen [10].

Este tipo de mutación, consiste en la alteración aleatoria de cada gen perteneciente al cromosoma, normalmente con una probabilidad pequeña de cambio definida al inicio del proceso evolutivo. Como es regular en esta clase de algoritmos, la probabilidad de mutación se define de forma estática a lo largo del proceso evolutivo. Para esta implementación la probabilidad de mutación inicia con un valor alto que durante x generaciones decae su valor en un factor γ hasta alcanzar un valor de $pm=\frac{1}{P}$ (donde P es el tamaño del cromosoma).

En el Algoritmo 3. Se puede apreciar el hecho que cuando se modifica uno de los genes dentro del cromosoma, si la configuración resultante, no es una solución admisible o valida, se prueba con otro gen que no sea el gen que se encontraba originalmente en dicha posición, si ocurre el caso de que ningún gen provoca una solución factible, el gen original retoma su antiguo lugar en el cromosoma, continuando así, hasta que se hayan probado todos y cada uno de los genes en el cromosoma. Si el proceso reviso cada uno de los cromosomas y no existió un cambio, el proceso vuelve a empezar y solo termina cuando por lo menos alguna posición es modificada.

Algoritmo 3 Mutación Multigen

Require: *Individuo p, Probabilidad de mutación - p_mut*

```
Direcciones = List(No. Direcciones ∈ R^n)
changed=0
while True do
    for i = 0 : i < longitud p do
      if Random ≤ p_mut then
          Original=p[i]
          shuffle(Direcciones)
          for j=0 : j < No.Direcciones. AND valido= False do
              if Direcciones[j] != Original then
                  p[i]=Direcciones[j]
                  Update_coordinates(p)  Actualizar coordenadas internas
                  valido= isValid(p)
          if No valido then
              p[i]=Original
              Update_coordinates(p)
          else
              changed=1
    if changed= =1 then
        BREAK
return True
```

3.1.4. Mutación *Pull-Moves Neighbors*

Siguiendo con la idea planteada por el método de *Path Relinking*, sobre generar varias soluciones candidatas. En este trabajo se presenta una versión *hill-climbing* del operador *Pull-Moves* introducido en [38]. Con la finalidad de explicar esta implementación primero se explicara a continuación el método de *Pull-Moves*.

El procedimiento empieza escogiendo un residuo i al tiempo t con coordenadas $(X_i(t), Y_i(t))$, considerando una retícula en 2D dimensiones. Suponemos que existe un posición libre L, adyacente a $i + 1$ en la posición $(X_{i+1}(t), Y_{i+1}(t))$ y diagonal adyacente a i (Horizontal o Vertical), además un posición C, adyacente mutuamente a L e i, podemos entonces comenzar a aplicar el método *Pull-Moves*, con las 4 posiciones

conocidas, las cuales dan la ilusión de formar un cuadro como se ve en la Figura 3.3a, un punto que hay aclarar es para que el método *Pull-Moves* ocurra, la posición *C* debe estar libre o ser estrictamente igual a $i-1$ en las coordenadas $(X_{i-1}(t),\ Y_{i-1}(t))$.

El caso más simple ocurre cuando *C* se encuentra posicionado en $i-1$, por lo tanto el movimiento sólo consiste en mover el residuo *i* hacia la posición *L*. Generando una nueva conformación al tiempo $t+1$ donde el residuo *i* se posiciona en $(X_i(t+1), Y_i(t+1))$. Este movimiento queda ilustrado en la Figura 3.3a. Cuando *C* se encuentra sobre una posición libre, Figura 3.3b, el residuo *i* se mueve hacia *L* y el nodo anterior $i-1$ se mueve a *C*, si el residuo $i-2$ quedo adyacente a *C*, el método de *Pull-Moves* queda completo.

Si el movimiento deja al residuo $i-2$ en una posición no adyacente a *C*. El proceso continua hasta que una configuración válida se encuentre, siguiendo con la notación propuesta por Lesh et al. Donde $j = i-2$, el residuo *j* toma la posición $(X_j(t+1), Y_j(t+1)) = (X_{j+2}(t), Y_{j+2}(t))$, y así sucesivamente hasta llegar al residuo 1, o a una configuración valida, tal y como se aprecia en la Figura 3.3c.

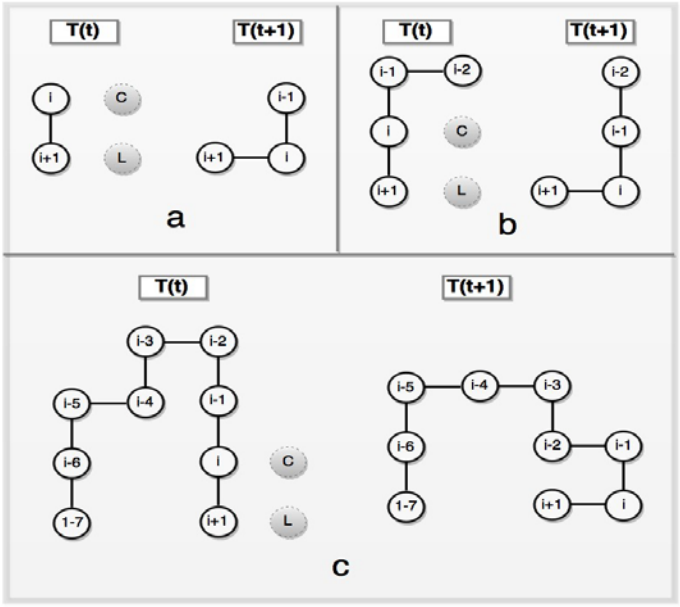

Figura 3.3: Esta figura ha sido tomada de [38], con el propósito de aclarar la funcionalidad del método de *Pull-Moves*.

Entonces, podemos describir el método de *Pull-Moves* como halar desde un residuo (aminoácido) *i* hasta el residuo 1, si fuese necesario el caso, para generar una nueva solución válida. No obstante, trabajar en la dirección contraria y desembocar en el último residuo, es igualmente válido. Teniendo esto en cuenta se implementó un mecanismo de *hill-climbing* sobre el operador de *Pull-Moves*, el cual considera todas las

posiciones libres *L* que un residuo *i* pueda tener ya sea que se tenga como base el residuo $i + 1$ o $i - 1$, tal y como se ilustra en la Figura 3.4. Por lo tanto, resumiendo podemos suponer que dada una secuencia de aminoácidos, el método *Pull-Moves* se aplica por cada residuo, sobre cada *L* que este pueda tener, generando de esta manera un conjunto de soluciones candidatas.

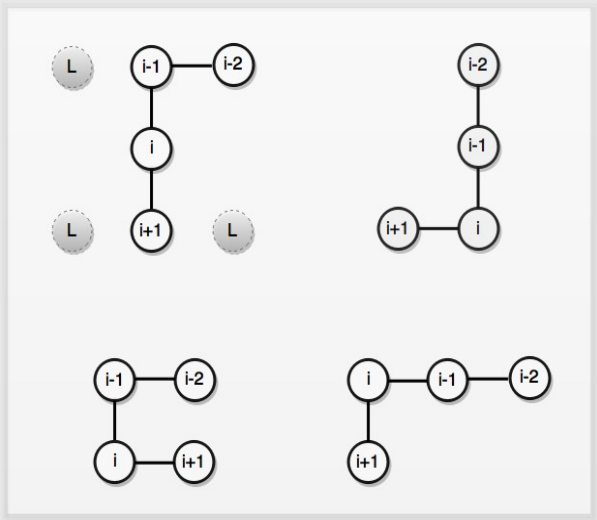

Figura 3.4: Esta figura ha sido tomada de [38], con el propósito de aclarar la funcionalidad del método de *Pull-Moves*.

Cuando Lesh et al. introdujo el método *Pull-Moves* expresó que este puede ser generalizado a un espacio en 3D. Esto es fácil de ver, si imaginamos un residuo *i* en el centro de un plano, las posiciones *L* y *C* para dicho residuo serán colocadas a lo largo del plano junto con el siguiente residuo para formar las esquinas de un cuadrado, para generalizar el proceso a un espacio en 3D, el residuo *i*, será posicionado en el centro de no uno, sino tres planos ortogonales, y las posiciones *L* y *C* pueden ser colocados sobre cualquiera de estos, siempre y cuando sea en el mismo.

3.1.5. Variable *Neighborhood Search*

Los métodos de búsqueda local para problemas de optimización, buscan mediante una serie de cambios locales sobre una solución inicial, mejorar a través del tiempo con la esperanza de encontrarse en un óptimo global. Aprovechamos la capacidad que poseen para explorar con más profundidad el espacio de búsqueda, en este trabajo se implementó un VNS (*variable neighborhood search*), cuya principal característica es su habilidad para visitar vecindarios cada vez más distantes para una actual solución.

El algoritmo básico del VNS, descrito en el Algoritmo 4, define N_k $(k = 1, \ldots k_{max})$, como un conjunto

finito de vecindarios, donde $N_{(k)}(S)$ representa el conjunto de soluciones pertenecientes al k-ésimo vecindario de una solución s. La mayoría de este tipo de métodos sólo trabaja con una estructura de vecindad, i.e $k_{max} = 1$. Si observamos a detalle el Algoritmo 4, podemos darnos cuenta que su forma de trabajar, es fácilmente adaptable al problema actual. Si bien, es sólo cuestión de implementar los mecanismos adecuados para cada etapa del algoritmo. En el caso del paso *4.a.* , podemos interpretar el generar una solución s' aleatoria de algunos de los vecindarios de s, como simplemente hacer una perturbación en s para conseguir s', y como hacemos dicha perturbación, pues para este paso se decidió utilizar el operador de mutación multigen, definido en la subsección 3.1.3,

Algoritmo 4 VNS

Require: initial solution s

1: **procedure** VNS
2: *top*:
3: $k \leftarrow 1$
4: *loop*: *stopping condition*:
5: *until* $k = k_{max}$
6: *4.a. Shaking* : *Generar un punto s' de forma aleatoria del k-ésimo vecino de s* $(s' \in N_{(k)}(s))$
7: *4.b. Local search* : *aplicar algún método de búsqueda local, con s' como solución inicial; teniendo s" como el óptimo local encontrado.*
8: *4.c. Move or not: Si s" es mejor que s, moverse* $(x \leftarrow s'')$*, y continuar la búsqueda con* $N_1(k \leftarrow 1)$
En otro caso $k \leftarrow k + 1$

Entonces la idea general de este algoritmo es dada una solución titular *s*, generar una solución s' proveniente de alguno vecindario, y aplicar una búsqueda local sobre s', para obtener un óptimo local s''. Si s'' es mejor que *s*, entonces s'' se convierte ahora en la solución titular; en otro caso un nuevo vecindario es utilizado para intentar mejorar *s*. Este proceso se repite hasta que la solución *s* no pueda ser mejorada.

3.1.6. Selección de Padres

La importancia de un mecanismo de selección de padres, es la de distinguir con base en su calidad, entre los individuos pertenecientes a una población, la idea es buscar aquellos individuos que serán padres para la siguiente generación. Los individuos con un alto valor de aptitud tienen una gran posibilidad para convertirse en padres, no obstante esto no quiere decir que aquellos individuos con una aptitud baja no tengan una probabilidad de ser elegidos como padres.

El operador de torneo descrito en el Algoritmo 5, tiene la propiedad de que no requiere ninguna información global acerca de la población. En lugar de eso, sólo requiere comparar dos individuos. El proceso inicia generando dos copias desordenadas de una población de tamaño *k*. Continuando se dispone a realizar una comparativa entre los individuos pertenecientes a dichas copias. El torneo entre dos individuos se realiza utilizando la dominación de *Pareto*, en cuyo caso de que sean *incomparables*, las soluciones son comparadas con base en su valor de *crowding-distance*. En cuyo caso de dicho valor sea igual, quiere decir que ambas soluciones son igual, por lo que el ganador del torneo se elige de forma aleatoria. El Algoritmo 6 muestra el funcionamiento del torneo.

Algoritmo 5 Selección por Torneo

Require: Población, k

```
1:  procedure SELECCIÓN TORNEO
2:   Individuals1 = (Población, k)–Desordena la población
3:   Individuals2 = (Población, k)–Desordena la población
4:    chosen-Arreglo de ganadores
5:    for i in (0,k,4) do
6:        chosen.add(Torneo(Individuals1_i, Individuals1_{i+1}))
7:        chosen.add(Torneo(Individuals1_{i+2}, Individuals1_{i+3}))
8:        chosen.add(Torneo(Individuals2_i, Individuals2_{i+1}))
9:        chosen.add(Torneo(Individuals2_{i+2}, Individuals2_{i+3}))
10:    return chosen
```

Algoritmo 6 Torneo

Require: ind1,ind2

```
1:  procedure TORNEO
2:      if ind1 ≺ ind2 then
3:          return ind1
4:      else if ind2 ≺ ind1 then
5:          return ind2
6:      if ind1.crowdingdistance < ind2.crowdingdistance then
7:          return ind2
8:      else if ind1.crowdingdistance > ind2.crowdingdistance then
9:          return ind1
10:     if random() ≤ 0.5 then
11:         return ind1
12:     return ind2
```

3.1.7. Selección de supervivientes y mecanismo de diversidad

Crowding-Distance

Mientras se converge al conjunto óptimo de Pareto, lo que se desea además es mantener un buen conjunto de soluciones que se propaguen por todo el espacio de búsqueda, manteniendo una buena diversidad entre los miembros de una población. Como se mencionó anteriormente el NSGA-II utiliza una métrica que estima la densidad de la población. Para obtener el valor de una solución, se requiere calcular la distancia promedio entre las dos soluciones continuas a la actual, el cálculo se realiza por cada objetivo y una vez obtenido cada distancia por objetivo, las distancias son sumadas para tener un único valor de *crowding-distance*. Esta métrica favorece a los individuos colocados en la posición de los extremos, puesto que considera que estas soluciones son las más alejadas a las demás. En la 3.5, la *crowding-distance* de la i-ésima solución es igual a la longitud promedio del lado del cuadrado.

Para calcular esta métrica se requiere ordenar la población acorde a cada función objetivo de forma ascendente. Al realizar este ordenamiento, en los extremos tendremos las soluciones con la mejor y peor aptitud para una determinada función objetivo, a las cuales se les asigna un valor de distancia de infinito, mientras que al resto se les asigna un valor de distancia normalizado de la diferencia entre los valores de las dos soluciones adyacentes. El proceso como se ve en el Algoritmo 7 continuá con las siguientes funciones

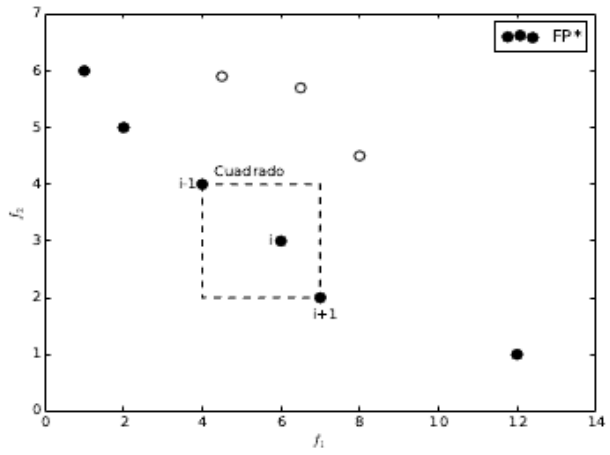

Figura 3.5: Cálculo de *crowding-distance* [14].

objetivo, ordenándolas con base en la función actual.

Algoritmo 7 Cálculo de crowding-distance

Require: Población P

1: **procedure** CROWDING-DISTANCE
2: l ← |P| *Número de soluciones en P*
3: **for** $p \in P$ **do**
4: p.distance = 0 *Asignar a cada p un valor de 0*
5: **for** $m \in Objetivos$ **do**
6: P ← sort(P,m) *Ordenar la población con base en el objetivo m*
7: P[1].distance ← P[l].distance ← ∞
8: **for** $i = 2 \rightarrow (l-1)$ **do**
9: P[i].distance ← P[i].distance + (P[i+1].m - P[i-1].m) /(f_m^{max} - f_m^{min})

FingerPrint

A diferencia de la métrica *Crowding-Distance* donde se determina el parecido de los individuos con base en su función de aptitud, el método de *fingerprint* [8, 26] lo determina basándose en la similitud de los contactos topológicos que puedan ocurrir dentro de las soluciones.

La métrica calcula un vector por cada individuo, donde cada componente *i* representa la distancia *manhattan* entre un par de aminoácidos (Línea 5 del algoritmo 8). Tal vector tiene un tamaño definido por $\binom{A}{2} - 2A + 3$ donde *A* representa el tamaño de la cadena proteica [26]. Por lo que sólo se contemplan los

aminoácidos cuya distancia entre ellos sea $|j\text{-}i| \geq 3$, es decir se calcula una distancia entre el primer y el cuarto aminoácido, primero y quinto y así sucesivamente hasta calcular la distancia entre el primero y el último, luego se realiza con el segundo y el quinto, continuado con cada uno hasta llegar al aminoácido *A* -3. Finalmente la diversidad de una solución, es la suma de calcular una distancia *hamming* entre el vector de la solución actual contra cada uno de los vectores restantes en la población (línea 10 del Algoritmo 8), donde dicha suma resultante se normaliza entre el número de individuos en la población (línea 11 del Algoritmo 8).

Algoritmo 8 *FingerPrint*

Require: Población, P

```
1:  procedure FINGERPRINT
2:      Popvectors ← []  Arreglo de vectores para cada individuo en la población
3:      for i in len(P) do
4:          distancia manhattan entre los aminoácidos de un individuo
5:          vector = ManhattanDistance(P_i)
6:          Popvectors.add(vector)   se añaden el vector de un  individuo.
7:      for P_i ∈ P do
8:          copyV ector = Popvectorscopia del arreglo de  vectores
9:          copyContacs.pop(i)Se saca el vector del individuo  i
10:         P_i.FingerPrint + = hammingDistance(Popvectors[i], copyV ector))
11:         P_i.FingerPrint = P_i.FingerPrint/len(P )
```

La métrica de fingerprint, es utilizada para estimar la densidad en una población de individuos, en este trabajo se utiliza como mecanismo de supervivencia. Para determinar que individuos en la población sobrevive, el mecanismo realiza una promedio entre los valores de *FingerPrint* de cada individuo, el cual dicho valor de diversidad se compara contra un umbral definido por el tomador de decisiones, si el valor de diversidad es menor que dicho umbral, se realiza un filtro en la población para identificar aquellos individuos cuyo valor de fingerprint se menor al umbral, de la población filtro sólo se selecciona un porcentaje (definido por el usuario) que serán eliminados para ser remplazados con nuevas soluciones generadas de forma aleatoria. El procedimiento de reemplazo queda definido en el Algoritmo 9.

A diferencia de la *Crowding-Distance* donde se realiza cada generación, el cálculo de diversidad fingerprint se realiza cada cierto número de generaciones, el cual al igual que el umbral y el porcentaje de reemplazo queda definido por el tomador de decisiones.

Algoritmo 9 Low-Diversity

Require: Población, Diversidad, porcentaje, umbral

```
1: procedure LOW-DIVERSITY
2:     if Diversidad<umbral then
3:         filterPop → población con los individuos menos diversos
4:         filterPop ← filter(P_i ∈ Population, where(P_i.fingerprint < threshold))
5:         filterPop ← filterPop[percentage]   conserva un porcentaje de los individuos
6:         replacePop ← X_1, X_2, ..., X_len(filterPop)   población generada aleatoria
7:         Fitnesses ← F_replacePop   Evaluar los individuos de replacePop
8:         for x in filterPop do
9:             filterPop[x] ← replacePop[x]
```

Reemplazo basado en edad

La básica idea de seleccionar supervivientes con base en la edad, es el hecho de no necesitar un valor de aptitud para determinar cuál de los individuos continuaran viviendo para la siguiente generación. El funcionamiento básico consiste en darle a cada individuo un esperanza de vida definido por el usuario, al cumplirse la edad son remplazados por nuevos individuos a excepción de la solución que encabeza el frente de Pareto.

4

Resultados

En esta sección se muestran las ejecuciones del algoritmo NSGA-II para el problema de plegamiento de proteínas para el modelo *HP* , usando las instancias estándar para el modelo *HP* mostradas en la Tabla 2.1 del Capítulo 2. Los experimentos se realizaron con población tamaño $d = 800$, cada individuo tiene una vida útil de 15 generaciones, a menos que sea la mejor solución en cuyo caso su edad no aumenta hasta que es desplazado de su posición como el mejor lugar. Con respecto al reemplazo utilizado el estimador de densidad *Fingerprint* se realiza cada 50 generaciones. El número de generaciones de cada instancia se muestra en la Tabla 4.1, los frentes de Pareto resultantes se normalizaron (suma de los objetivos), para poder hacer una comparativa con respecto a la literatura de los resultados mono-objetivo, todos los resultados experimentales reportados en esta sección son el promedio de 30 ejecuciones independientes.

Tabla 4.1: Tabla de generaciones.

Instancias	Número de Generaciones
2D 1-10, 3D 1-4	5000
2D 11-15, 3D 5-12	3000

Con lo que respecta a la metaheurística implementada internamente en el algoritmo NSGA-II. El VNS quedó configurado con un máximo número de 1000 evaluaciones y sólo se aplica a los siguientes individuos en la población:

- Población Descendiente: Aquel con menor aptitud.
- Población Descendiente: Aquel con la mejor aptitud.

- Población Descendiente: Uno aleatorio entre los mejores de los hijos, que no sea el mejor de ellos.
- Población Descendiente: Uno aleatorio entre los mejores de los hijos, que no sea el mejor de ellos.
- Población Padre: El mejor tercero entre los padres.
- Población Padre: El mejor cuarto entre los padres.

Se marca una diferencia entre los individuos de las poblaciones puesto que el algoritmo VNS se aplica antes de realizar antes de que las poblaciones se unan en un sola gran población, tal y como se ve en el algoritmo

Entonces recapitulando, todos los parámetros que se utilizaron para las ejecuciones podemos verlos claramente en la Tabla 4.2.

Tabla 4.2: Tabla de Parámetros.

Parámetro	Asignación
Tamaño de Población	800
Probabilidad de cruza	0.9
Probabilidad inicial de mutación	0.3
Decremento Probabilidad mutación	0.005/10 Gen.
Esperanza de vida	15
FingerPrint	50 Gen.
Umbral	0.5
Porcentaje de reemplazo	20 %
VNS	1000 Eval.

4.1. Comparación entre las funciones de energía Multi-objetivo

La característica básica de cualquier metaheurística, es explorar eficientemente el espacio de búsqueda con el objetivo de encontrar soluciones próximas al óptimo global [5]. La idea de transformar una función mono-objetivo en términos de dos o más objetivos, introduce un cambio fundamental en la forma en la que se explora dicho espacio, mejorando potencialmente la eficiencia de la búsqueda.

Recordando las funciones multi-objetivo descritas en la subsección 2.1.1 del Capítulo 2. Dentro de [24] los autores mostraron una comparativa de estas funciones mostrando una clara superioridad la función *H-subset*, seguida por la función *Locality* y por último la función *Parity*

Después de un análisis comparativo entre las funciones *H-Subset* y *Locality* bajo el algoritmo planteado en esta investigación, podemos observar en la Tabla 4.3 que el comportamiento de ambas funciones es similar, puesto que los resultados mostrados, indican que se está encontrado el óptimo conocido, para la muestra de instancias seleccionadas para este experimento, las cuales son las instancias más pequeñas. Para este experimento el valor δ de la función *Locality* es igual a 7, mientras que la configuración para la función *H-Subset* define a que grupo pertenece cada aminoácido hidrofóbico desde el principio y lo mantiene así hasta el fin del algoritmo.

Tabla 4.3: Comparativa Funciones bajo el NSGA-II

			NSGA-II		NSGA-II	
			LD		HD	
Seq.	L	Best	β(f)	υ	β(f)	υ
2d1	18	-4	-4(100 %)	-4	-4(100 %)	-4
2d2	18	-8	-8(100 %)	-8	-8(100 %)	-8
2d3	18	-9	-9(100 %)	-9	-9(100 %)	-9
2d4	20	-9	-9(100 %)	-9	-9(100 %)	-9
2d5	20	-10	-10(100 %)	-10	-10(100 %)	-10
2d6	24	-9	-9(100 %)	-9	-9(100 %)	-9
2d7	25	-8	-8(100 %)	-8	-8(100 %)	-8
2d8	36	-14	-14(100 %)	-14	-14(100 %)	-14
2d9	48	-23	-23(100 %)	-23	-23(100 %)	-23

Best: β, Best frequency: β(f), Average: υ

LD: Locality, HD H-Subset

Con base en el tiempo computacional que el algoritmo requiere para encontrar una solución, se observó que la función *Locality* actuá de manera más rápida que la función *H-Subset*, tal y como se muestra en la Tabla 4.4, donde se muestra el tiempo promedio en segundos requerido por cada instancia. Aunque la función *Locality* trabaja más eficiente para la mayoría de las instancias, se tiene de excepción la instancia *2d7*, donde *H-Subset* es mejor. No obstante, conforme el tamaño de la cadena va aumentando se hace más notoria la

diferencia entre el tiempo promedio requerido por cada función de energía.

Tabla 4.4: Tiempo Promedio por Funciones

		LD	HD
Seq.	L	Tiempo	Tiempo
2d1	18	43.53	50.75
2d2	18	46.9	72.05
2d3	18	81.76	293.2
2d4	20	22.0	38.65
2d5	20	32.6	34.05
2d6	24	76.56	117.6
2d7	25	301.53	171.15
2d8	36	397.73	505.85
2d9	48	1285.1	82331.75

LD: Locality, HD H-Subset

Por lo tanto con base en los datos mostrados y la preferencia de evitar la aleatoriedad de los grupos, necesarios para la función *H-Subset*, se eligió la función *Locality*, como función objetivo principal, para evaluar las conformaciones presentadas por las instancias grandes en 2D y para cada una de las instancias presentadas para la versión 3D del modelo.

4.2. Resultados 2D

En esta sección se presentan los resultados obtenidos por el NSGA-II. Los cuales se muestran en la Tabla 4.5. Por cada instancia se reporta: El mejor óptimo encontrado (β), la tasa de éxito por cada óptimo conocido (β(f)) y el promedio obtenido por cada instancia (υ). Si se observa las columnas 3 y 4, podemos apreciar que el algoritmo logra encontrar el mejor óptimo conocido para las primeras 11 instancias. Mientras tanto para las instancias mayores la tasa de éxito es demasiado baja, además el óptimo encontrado, no es el mejor conocido para dichas instancia.

Para realizar una comparativa del estado del arte contra las soluciones obtenidas, se seleccionó los resultados propuestos, en el trabajo original de la función *Locality* en un algoritmo evolutivo *EA (1+1)* [27] puesto que es la función utilizada para esta fase, aparte una comparativa con los resultados propuestos por Chira et al. (ELF) [8], Cutello et al. (IA) [13], Islam et al. (CMA) [32] los cuales utilizaron su propia variante para la función de energía para el modelo, Santana et al. (EDA) [46] y por último Rego et al. (FF) [43].

Tabla 4.5: Resultados de las instancias estándar en 2D.

			NSGA-ll		EA(1+1)		ELF		IA		CMA		EDA		FF	
			LD		LD		SO		SO		I09		SO		SO	
Seq.	L	Best	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ
2d1	18	-4	-4(100 %)	-4	-4(03 %)	-2.69			-4(100 %)	-4						
2d2	18	-8	-8(100 %)	-8	-8(31 %)	-7.16			-8(100 %)	-8						
2d3	18	-9	-9(100 %)	-9	-9(02 %)	-7.39			-9(100 %)	-9						
2d4	20	-9	-9(100 %)	-9	-9(11 %)	-7.23	-9(100 %)	n/a	-9(100 %)	-9	-9(100 %)	-9	-9(100 %)	n/a	-9(n/a)	n/a
2d5	20	-10	-10(100 %)	-10	-9(1 %)	-7.06			-10(100 %)	-10						
2d6	24	-9	-9(100 %)	-9	-9(2 %)	-7.30	-9(100 %)	n/a	-9(100 %)	-9	-9(100 %)	-9	-9(100 %)	n/a	-9(n/a)	n/a
2d7	25	-8	-8(100 %)	-8	-8(7 %)	-6.17	-8(100 %)	n/a	-8(100 %)	-8	-8(100 %)	-8	-8(100 %)	n/a	-8(n/a)	n/a
2d8	36	-14	-14(100 %)	-14	-13(4 %)	-10.61	-14(68 %)	n/a	-14(100 %)	-14	-14(100 %)	-14	-14(16 %)	n/a	-14(n/a)	n/a
2d9	48	-23	-23(100 %)	-23	-20(2 %)	-16.29	-23(12 %)	n/a	-23(56 %)	-22.47	-23(n/a)	-22.87	-23(18 %)	n/a	-23(n/a)	n/a
2d10	50	-21	-21(100 %)	-21	-19(1 %)	-15.07	-21(60 %)	n/a	-21(100 %)	-21	-21(100 %)	-21	-21(98 %)	n/a	-21(n/a)	n/a
2d11	60	-36	-35(100 %)	-35	-30(1 %)	-15.07	-35(12 %)	n/a	-35(100 %)	-35	-36(n/a)	-35.1	-35(18 %)	n/a	-36(n/a)	n/a
2d12	64	-42	-42(10 %)	-39.3	-30(4 %)	-27.80	-40(8 %)	n/a	-42(10 %)	-39.3	-42(n/a)	-39.6	-42(12 %)	n/a	-42(n/a)	n/a
2d13	85	-53	-52(50 %)	-51.5	-44(1 %)	-26.61					-53(n/a)	-50.91	-52(4 %)	n/a	-53(n/a)	n/a
2d14	100	-48	-46(40 %)	-45.1	-39(2 %)	-38.09					-47(n/a)	-45.41	-47(2 %)	n/a	-48(n/a)	n/a
2d15	100	-50	-48(5 %)	-45.9	-39(7 %)	-34.41					-50(n/a)	-47.29	-48(2 %)	n/a	-50(n/a)	n/a

Best: β, Best frequency: β(f), Average: υ

LD=Locality , SO= *Single Objective* ([17]), I09 = Función Islam. [32]

En primer lugar se examina como el algoritmo NSGA-II especialmente diseñado para problemas multi-objetivo, utilizando la función *Locality* supera los resultados mostrados por el algoritmo EA(1+1) el cual utiliza la misma función objetivo. Aunque recordando que el objetivo principal de ese trabajo era presentar una versión multi-objetivo de la función de energía original de *Dill*, no buscaban más que sólo demostrar una mejor eficiencia contra está.

Por otro lado, respecto a los resultados recopilados en la literatura, el comportamiento del algoritmo se ve favorable para la mayoría de las instancias. En el caso de comparar los resultados contra el algoritmo ELF, los resultados muestran un mayor excitó en la ocurrencia del óptimo conocido, desde la instancia 8 en adelante, siendo el NSGA-II superior para estas instancias. Por otra parte la comparación contra el *IA*, contemplamos que los resultados son iguales para todas las instancias probadas en el *IA*, sólo siendo mejor en la instancia 9. Contraste a los algoritmos *CMA* y *EDA*, para el *CMA* no se reportan tasas de excitó, no obstante, con base en los promedios reportados la tasa de éxito no es un 100 % para todas ellas, aun así, el algoritmo logro encontrar el óptimo conocido superando en comparación a los resultados del NSGA-II, con lo que respecta a los *EDA*, en la Tabla 4.5 se aprecia que el promedio no se encuentra disponible, comprobamos que para las instancias no en todos los casos de prueba se logro encontrar el óptimo conocido, a pesar de eso los *EDA* proporcionan mejores resultados para las instancias mayores. Para el último algoritmo, el *FF* tenemos que en comparación con el NSGA-II logra encontrar el óptimo conocido para cada instancia.

Si bien el algoritmo NSGA-II queda atrás para la mayoría de las instancias grandes, con esta configuración de parámetros, Para futuros trabajos un mayor número de población o generaciones ayudaría a mejorar la eficiencia del algoritmo.

En cuestión de eficiencia el algoritmo NSGA-II fue diseñado en el lenguaje de programación *python* 2.7, ejecutado en un procesador Intel(R) Xeon(R) X5660 a 2.80GHz, el tiempo computacional requerido por cada instancia se reporta en la Tabla 4.6

Tabla 4.6: Comparación de algoritmos - Eficiencia

Instancias		Tiempo en segundo		
Seq.	L	NSGA-II	FF	REMC
2d1	18	43.5	n/a	n/a
2d2	18	46.9	n/a	n/a
2d3	18	81.76	n/a	n/a
2d4	20	22.0	1	1
2d5	20	32.6	n/a	n/a
2d6	24	76.56	1	1
2d7	25	301.53	1	1
2d8	36	397.73	2	1
2d9	48	1285.1	4	1
2d10	50	1612.43	11	1
2d11	60	1060837.5	23	13
2d12	64	707503.36	12	6
2d13	85	1079606.75	13	38
2d14	100	223214.0	30888	480
2d15	100	196652.95	32503	72

En comparación con dos de los algoritmos más eficientes presentados en la literatura el *FF* y el *REMC* [49], se ve como el algoritmo NSGA-II queda atrás respecto al tiempo que le toma encontrar las soluciones.

En Resumen el algoritmo NSGA-II demuestra que tiene el potencial para encontrar los óptimos conocidos para las instancias, con el inconveniente de requerir más tiempo computacional para llevar acabo la tarea.

4.3. Resultados 3D

Para esta sección se muestran los resultados del NSGA-II para el modelo en 3D. Los cuales se muestran en la Tabla 4.7. Al igual para los resultados en 2D se reportan : El mejor óptimo encontrado (β), la tasa de éxito por cada óptimo conocido (β(f)) y el promedio obtenido por cada instancia (υ). En la tabla podemos apreciar que el modelo logra una tasa de éxito del 100 % para las primeras cuatro instancias en 3D, descritas en 2.2, sin embargo para las siguientes dos instancias la tasa de éxito decae, mientras tanto para las instancias 7 en adelante el algoritmo no es capaz de dar con el óptimo conocido

Para la comparativa con el estado del arte, se toman los resultados mostrado en el trabajo original de la función *Locality* en un algoritmo *EA (1+1)* [27] , además se realiza una comparativa con los resultados propuestos por Chira et al. (ELF) [8], Cutello et al. (IA) [13], Islam et al. (CMA) [32] que igual para el modelo en 2D utiliza su propia variante para la función energía para el modelo, Santana et al. (EDA) [46], además para esta fase se agrega los resultados mostrados por lin et al. (PSO) [39], pues en este trabajo sólo se considera el modelo 3D y por último se toman los resultados de un algoritmo genético presentado por Bovskovic et al. (GA_{PSP}) [6]

Igual que con el modelo en 2D, se aprecia claramente como el NSGA-II utilizando la función *Locality* aumenta la tasa de éxito, que su contraparte con el algoritmo EA(1+1).

Respecto a los resultados de la literatura, el comportamiento del algoritmo se ve estable con sólo alguna de las instancia. Si comparamos los resultados presentados por el *IA* contra los del NSGA-II, se muestra una conducta mejor para el NSGA-II, por otra parte, en contraste con los resultados arrojados por el *CMA* mantienen el mismo comportamiento para las instancias pequeñas y supera al NSGA-II para las más grandes. Si se considera los algoritmos *EDA* y *PSO*, el NSGA-II supera los resultados mostrados por ambos algoritmos en cada una de las instancias que comparten. Por último contra el algoritmo GA_{PSP} , logra encontrar el óptimo conocido para cada una de las instancias, por lo tanto supera los resultados obtenidos en esta investigación.

Tabla 4.7: Resultados de las instancias estándar en 3D.

			NSGA-ll		EA(1+1)		IA		CMA		EDA		PSO		ttA_{psp}	
			LD		LD		SO		I09		SO		SO		SO	
Seq.	L	Best	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ	β(f)	υ
3d1	20	-11	-11(100 %)	-11	-11(94 %)	-10.94	-11(n/a)	-11	-11(100 %)	-11	-11(n/a)	-11	-11(n/a)	-11	-11(n/a)	-11
3d2	24	-13	-13(100 %)	-13	-13(66 %)	-12.53	-13(n/a)	-13	-13(100 %)	-13	-13(n/a)	-13	-13(n/a)	-13	-13(n/a)	-13
3d3	25	-9	-9(100 %)	-9	-9(95 %)	-8.95	-9(n/a)	-9	-9(100 %)	-9	-9(n/a)	-9	-9(n/a)	-9	-9(n/a)	-9
3d4	36	-18	-18(100 %)	-18	-18(46 %)	-16.97	-18(n/a)	-16.76	-18(100 %)	-18	-18(n/a)	-16.5	-18(n/a)	-17.72	-18(n/a)	-18
3d5	46	-35	-35(36.67 %)	-33.77	-31(1 %)	-27.53										
3d6	48	-31	-31(93.33 %)	-30.93	-31(1 %)	-26.66	-29(n/a)	-25.16	-31(100 %)	-31	-29(n/a)	-27.24	-29(n/a)	-28.88	-31(n/a)	-31
3d7	50	-34	-33(24 %)	-31.8	-28(1 %)	-24.31	-23(n/a)	-22.6	-31(100 %)	-31	-31(n/a)	-25.94	-26(n/a)	-25.92	-34(n/a)	-33.96
3d8	58	-44	-42(10 %)	-39.65	-36(2 %)	-31.98										
3d9	60	-55	-53(33.33 %)	-52.3	-47(3 %)	-42.88	-41(n/a)	-39.28	-54(80 %)	-52.52	-49(n/a)	-46.3	-49(n/a)	-48.62	-55(n/a)	-54.46
3d10	64	-59	-58(5 %)	-55.55	-50(1 %)	-43.29	-42(n/a)	-39.08	-58(65 %)	-56.3	-52(n/a)	-46.78			-59(n/a)	-59
3d11	67	-56	-48(5 %)	-45.6	-41(1 %)	-36.1										
3d12	88	-72	-60(10 %)	-57.2	-53(1 %)	-46.13										

Best: β, Best frequency: β(f), Average: υ

LD=Locality , SO= *Single Objective* ([17]), I09 = Función Islam. [32]

5

Conclusiones

El objetivo general de esta investigación es presentar un algoritmo diseñado especialmente para trabajar con una función mono-objetivo al ser trasformada a multi-objetivo, aplicado al problema de *HP* de proteínas. A continuación se detalla un resumen de los resultados obtenidos, para alcanzar cada uno de los objetivos específicos planteados.

1. Analizar el comportamiento de las funciones propuestas en la literatura para el modelo *HP*.

En la sección 4.1 del Capítulo 4, se presenta una discusión acerca de las diferentes funciones de energía multi-objetivo propuestas para el modelo *HP* , obtenidas de una revisión de la literatura. De esta revisión, se pudo apreciar el comportamiento de cómo cada función, pretende explorar el espacio de soluciones posibles. Además se presenta una comparativa de ellas con la finalidad de determinar cuál de ellas abunda mejor en el espacio factible de soluciones. Dentro de esta revisión podríamos concluir cuál de las funciones, sería la apta, para realizar en los experimentos futuros que se presentaron a lo largo de la investigación. Durante la fase de experimentación se concluyó que la mejor función de energía que interactuá mejor con el algoritmo planteado es la función de *Locality*

2. Desarrollar un algoritmo genético, apto para el problema. Para evolucionar potenciales soluciones al problema de predicción de la estructura terciaria de las proteínas, representadas bajo el modelo *HP* , se desarrolló un algoritmo evolutivo multi-objetivo, con un operador de cruza, dos operadores de mutación y un método de diversidad basado en interacciones hidrofóbicas. Se realizaron diferentes experimentos preliminares con instancias cortas para comprobar la utilidad de los operadores de variación y el método de diversidad. Con base estos resultados se obtuvieron diferentes datos estadísticos, con la finalidad de determinar cuál del conjunto de parámetros, era el conjunto que proporciona resultados cercanos a los deseados. Una vez terminados los experimentos preliminares se concluyó que los parámetros descritos en la tabla 4.2 eran los mejores para los experimentos definitivos de la investigación.

3. Competir con el grado de discriminación de las funciones multi-objetivo propuestas en la literatura.

En este proceso culmina los procesos realizados en la investigación, con base en las conclusiones que se tomaron en los objetivos anteriores se desarrollaron las últimas pruebas, con la finalidad de que estas lograran cumplir el objetivo general.

”Aplicando un análisis multi-objetivo al problema *HP* de proteínas, podremos predecir la conformación nativa de una proteína", si observamos los resultados, propuestos por el algoritmo presentado en esta investigación se puede observar claramente, que la utilización de un algoritmo apropiado para tratar con problemas multi-objetivo, realiza una buena exploración en el espacio de conformaciones. Si bien es claro que los resultados son competitivos respecto a los mostrados en la literatura, estando sólo por debajo para algunos algoritmos, indica que este podría ser un buen camino a seguir para tratar con el problema. La utilización de este tipo de técnicas podrían ayudar fuertemente al problema de replegado de proteínas, siendo una ayuda para conocer las conformaciones nativas, no obstante, como trabajo futuro se puede seguir explorando el área de algoritmos multi-objetivo para tratar a este problema y en el peculiar caso de esta investigación realizar más pruebas, utilizando más población, más generaciones o incluso trabajar con un modelo de islas para el manejo de las poblaciones en el algoritmo.

Referencias

[1] Cedric Avanthay, Alain Hertz, and Nicolas Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[2] Ugo Bastolla, Helge Frauenkron, Erwin Gerstner, Peter Grassberger, and Walter Nadler. Testing a new monte carlo algorithm for protein folding. *arXiv preprint cond-mat/9710030*, 1997.

[3] Igor Berenboym and Mireille Avigal. Genetic algorithms with local search optimization for protein structure prediction problem. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1097–1098. ACM, 2008.

[4] Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (hp) model is np-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.

[5] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[6] Borko Boškovic´ and Janez Brest. Genetic algorithm with advanced mechanisms applied to the protein structure prediction in a hydrophobic-polar model and cubic lattice. *Applied Soft Computing*, 45:61–70, 2016.

[7] David Arboledas Brihuega. *Jerarquía estructural de las proteínas*. Editorial Club Universitario, 2011.

[8] Camelia Chira. A hybrid evolutionary approach to protein structure prediction with lattice models. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2300–2306. IEEE, 2011.

[9] Carlos A Coello Coello and Col San Pedro Zacatenco. Introducción a la computación evolutiva (notas de curso). *CINVESTAV-IPN, Departamento de Ingeniería Eléctrica, Sección de Computación. México, DF*, 2004.

[10] Carlos A Coello Coello and Col San Pedro Zacatenco. Introducción a la computación evolutiva (notas de curso). *CINVESTAV-IPN, Departamento de Ingeniería Eléctrica, Sección de Computación. México, DF*, 2015.

[11] Carlos Coello Coello, Gary B Lamont, and David A Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007.

[12] Pierluigi Crescenzi, Deborah Goldman, Christos Papadimitriou, Antonio Piccolboni, and Mihalis Yannakakis. On the complexity of protein folding. *Journal of computational biology*, 5(3):423–465, 1998.

[13] Vincenzo Cutello, Giuseppe Nicosia, Mario Pavone, and Jonathan Timmis. An immune algorithm for protein structure prediction on lattice models. *IEEE transactions on evolutionary computation*, 11(1):101–117, 2007.

[14] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[15] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[16] Thomas M Devlin. *Bioquímica: libro de texto con aplicaciones clínicas*. Reverté, 2004.

[17] Ken A Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, 1985.

[18] Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[19] Manfred Eigen. *Ingo Rechenberg evolution strategy optimization of technical systems according to principles of biologishen Evolution*. afterword by Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt, 1973.

[20] Mario Garza Fabre. *Optimización de problemas con más de tres objetivos mediante algoritmos evolutivos*. PhD thesis, Master's thesis, Laboratorio de Tecnologıas de la Información, CINVESTAV-IPN, Cd. Victoria, Tamaulipas, México, 2009.

[21] E Feduchi, I Blasco, CS Romero, and E Yáñez. Bioquímica. conceptos esenciales. *Panamericana*, 2011.

[22] Lawrence J Fogel. *Artificial Intelligence Through Simulated Evolution.[By] Lawrence J. Fogel... Alvin J. Owens... Michael J. Walsh*. John Wiley & Sons, 1966.

[23] Mario Garza-Fabre, Eduardo Rodriguez-Tello, and Gregorio Toscano-Pulido. Comparing alternative energy functions for the hp model of protein structure prediction. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2307–2314. IEEE, 2011.

[24] Mario Garza-Fabre, Eduardo Rodriguez-Tello, and Gregorio Toscano-Pulido. An improved multiobjectivization strategy for hp model-based protein structure prediction. In *International Conference on Parallel Problem Solving from Nature*, pages 82–92. Springer, 2012.

[25] Mario Garza-Fabre, Eduardo Rodriguez-Tello, and Gregorio Toscano-Pulido. Multiobjectivizing the hp model for protein structure prediction. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 182–193. Springer, 2012.

[26] Mario Garza-Fabre, Eduardo Rodriguez-Tello, and Gregorio Toscano-Pulido. Constraint-handling through multi-objective optimization: The hydrophobic-polar model for protein structure prediction. *Computers & Operations Research*, 53:128–153, 2015.

[27] Mario Garza-Fabre, Gregorio Toscano-Pulido, and Eduardo Rodriguez-Tello. Locality-based multiobjectivization for the hp model of protein structure prediction. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 473–480. ACM, 2012.

[28] Fred Glover. A template for scatter search and path relinking. *Lecture notes in computer science*, 1363:13–54, 1998.

[29] David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989:102, 1989.

[30] Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467, 2001.

[31] K. E. Van hole. *Advances in Protein Chemistry*, volume 47. Academic Press, 1995.

[32] Md Kamrul Islam and Madhu Chetty. Clustered memetic algorithm with local heuristics for ab initio protein structure prediction. *IEEE Transactions on Evolutionary Computation*, 17(4):558–576, 2009.

[33] Holland John. Holland, adaptation in natural and artificial systems, 1992.

[34] Andrzej Kolinski and Jeffrey Skolnick. Reduced models of proteins and their applications. *Polymer*, 45(2):511–524, 2004.

[35] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[36] Natalio Krasnogor, BP Blackburne, Edmund K Burke, and Jonathan D Hirst. Multimeme algorithms for protein structure prediction. In *International Conference on Parallel Problem Solving from Nature*, pages 769–778. Springer, 2002.

[37] Natalio Krasnogor, William E Hart, Jim Smith, and David A Pelta. Protein structure prediction with evolutionary algorithms. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pages 1596–1601. Morgan Kaufmann Publishers Inc., 1999.

[38] Neal Lesh, Michael Mitzenmacher, and Sue Whitesides. A complete and effective move set for simplified protein folding. In *Proceedings of the seventh annual international conference on Research in computational molecular biology*, pages 188–195. ACM, 2003.

[39] Cheng-Jian Lin and Shih-Chieh Su. Protein 3 d hp model folding simulation using a hybrid of genetic algorithm and particle swarm optimization. *International Journal of Fuzzy Systems*, 13(2):140–147, 2011.

[40] Nenad Mladenović, Pierre Hansen, and D V. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[41] R Baños Navarro. *Meta heurísticas Híbridas para Optimización Mono-objetivo y Multi-objetivo*. PhD thesis, Tesis Doctoral, Almería, Spain, December 2006. Available at www. ace. ual. es/˜ rbanos/CV. html, last date of access May 12 2009, 2009.

[42] Víctor Hernán Arcila Quiceno. Bio-informática un campo por conocer. *REDVET. Revista Electrónica de Veterinaria*, 7(11):1–9, 2006.

[43] César Rego, Haitao Li, and Fred Glover. A filter-and-fan approach to the 2d hp model of the protein folding problem. *Annals of Operations Research*, 188(1):389–414, 2011.

[44] Mauricio GC Resende, Celso C Ribeiro, Fred Glover, and Rafael Martí. Scatter search and path-relinking: Fundamentals, advances, and applications. In *Handbook of metaheuristics*, pages 87–107. Springer, 2010.

[45] Piedad Tolmos Rodríguez-Piñero. *Introducción a los algoritmos genéticos y sus aplicaciones*. Universidad Rey Juan Carlos, Servicio de Publicaciones, 2003.

[46] Roberto Santana, Pedro Larrañaga, and Jose A Lozano. Protein folding in simplified models with estimation of distribution algorithms. *IEEE transactions on Evolutionary Computation*, 12(4):418–438, 2008.

[47] Alena Shmygelska and Holger H Hoos. An improved ant colony optimisation algorithm for the 2d hp protein folding problem. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 400–417. Springer, 2003.

[48] SN Sivanandam and SN Deepa. *Introduction to genetic algorithms*. Springer Science & Business Media, 2007.

[49] Chris Thachuk, Alena Shmygelska, and Holger H Hoos. A replica exchange monte carlo algorithm for protein folding in the hp model. *BMC bioinformatics*, 8(1):1, 2007.

[50] José Luis Urdiales. Tema 4 relación entre la estructura y la función, jan 2015.

[51] David A Van Veldhuizen. Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, DTIC Document, 1999.

[52] Peter H Verdier and WH Stockmayer. Monte carlo calculations on the dynamics of polymers in dilute solution. *The Journal of Chemical Physics*, 36(1):227–235, 1962.

[53] Katherine Viteri, Christian Salazar, Carlos Paredes, and J Muñoz. Algoritmos genéticos.

[54] Donald Voet and Judith G Voet. *Bioquímica*. Ed. Médica Panamericana, 2006.

[55] Xinjie Yu and Mitsuo Gen. *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.