ALGORITMOS HEUR´ISTICOS EN BIOINFORMÁTICA

TESIS DOCTORAL

David Alejandro Pelta
dpelta@ugr.es

DIRECTORES: José L. Verdegay - Armando Blanco

Depto. de Ciencias de la Computación e
Inteligencia Artificial.
Universidad de Granada, 18071 Granada

Índice General

Agradecimientos **7**

Introducción **11**

1 Optimización Combinatoria y Metaheurísticas **17**

1.1 Definiciones Preliminares 18

1.2 Propiedades de las Metaheurísticas 20

1.3 Ejemplos de Metaheurísticas 22

2 *FANS*: una Heurística para Problemas de Optimización **33**

2.1 Presentación de *FANS* 35

2.1.1 El Operador de Modificación 36

2.1.2 La Valoración Difusa 37

2.1.3 El Administrador de Operación 37

2.1.4 El Administrador de Vecindario 38

2.1.5 Parámetros Globales 39

2.1.6 Manejo de optimos locales 39

2.1.7 El Algoritmo 41

2.2 La Valoración Difusa como Mecanismo de Variación del Comportamiento de *FANS* 42

2.3 Comentarios Sobre la Implementación 46

2.4 Utilización de Múltiples Operadores en el Proceso de Búsqueda 48

2.4.1 "Un Operador, un Landscape" 48

2.4.2 Justificación Experimental 52
2.5 Conclusiones 56

3 Verificación Experimental de *FANS* 59
3.1 *FANS* vs *AG* en Problemas de Mochila 60
3.1.1 Formulación de los Problemas de Mochila 61
3.1.2 Implementación de *FANS* 62
3.1.3 Descripción del Algoritmo Genético 66
3.2 Problemas de Mochila Estandard 67
3.2.1 Instancias de Prueba 67
3.2.2 Experimentos y Resultados 68
3.3 Mochila con Múltiples restricciones 75
3.3.1 Instancias de Prueba 75
3.3.2 Experimentos y Resultados 75
3.4 Experimentos sobre Funciones Reales 80
3.4.1 Funciones de Prueba 80
3.4.2 Implementación de *FANS* 81
3.4.3 Experimentos y Resultados 84
3.5 Análisis Complementarios de *FANS* 89
3.5.1 Análisis de la Calidad de las Soluciones Investigadas . . 89
3.5.2 Evaluación de Administradores de Vecindarios 93
3.6 Conclusiones 99

4 Aplicación de *FANS* a Problemas de Bioinformática 101
4.1 Conceptos Básicos 103
4.2 El Problema de Predicción de Estructura 108
4.2.1 Problemas en la determinación de la estructura 110
4.2.2 Modelizaciones del Problema 112
4.3 El Problema de Comparación de Estructuras 117
4.3.1 Consideraciones Generales 118
4.3.2 Comparación de Estructuras vía Matrices de Distancia . 120
4.3.3 Comparación de Estructuras vía Cliques 121

4.3.4 Comparación de Estructuras vía Superposición de Mapas de Contacto 122
4.4 *FANS* para el Problema de Predicción de Estructura 123
4.4.1 Introducción 123
4.4.2 Diseño de Algoritmos para *PSP* 125
4.4.3 Influencia de la Codificación en un *AG* 130
4.4.4 Influencia de la Codificación en *FANS* 133
4.4.5 Evitando los problemas del *AG* en *PSP* 136
4.4.6 Instancias de prueba utilizadas en los experimentos . . . 139
4.5 *FANS* para el Problema de Emparejamiento Estructural de Moléculas 141
4.5.1 Definición del Problema 142
4.5.2 Implementación de *FANS* para *MSM* 142
4.5.3 Experimentos y Resultados 145
4.6 Conclusiones 155

Conclusiones y Trabajos Futuros 156

Bibliografía 167

Agradecimientos

La finalización de esta tesis marca el fin de una etapa y el comienzo de otra. El camino no ha sido fácil y obviamente no hubiera podido recorrerlo solo y estando a más de 10000 km. de mi país y de los míos.

Es por eso que en estos párrafos quiero dejar expresado mi agradecimiento a todos aquellos que contribuyeron, no sólo en el ámbito académico, a que estos años de doctorado y estadía en Granada, hayan sido más que gratificantes y dignos de recuerdo.

En primer lugar, es practicamente imposible para cualquier latinoamericano realizar un doctorado en Europa sin apoyo económico, así que deseo agradecer a los Prof. Nicolás Perez de la Blanca, Armando de Giusti y Gabriel Baum por su ayuda mediante una beca del proyecto Cometas del Programa ALFA. Sin este apoyo inicial, quizás nunca hubiera podido iniciar el doctorado. Posteriormente, mi beca del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) de Argentina me permitió llegar a este punto. Agradezco a Gabriel Baum la confianza y el haber aceptado ser mi director desde Argentina.

Deseo agradecer profundamente a mis directores José Luis Verdegay y Armando Blanco. Podría hacer una lista de cosas por las cuales siento que les debo un agradecimiento, pero la resumo en tres cosas: gracias por la constante disponibilidad, apoyo y también su amistad. En estos casi 4 años de trabajo conjunto he aprendido mucho y creo que más que una relación tutor-alumno (o jefe-esclavo!), en mi opinión hemos sido verdaderos compañeros de trabajo. Considerando que antes de empezar no nos conocíamos y viendo donde estamos ahora, solo me queda expresar el deseo que ojalá sigamos trabajando juntos.

También vaya mi agradecimiento a todo el Departamento de Ciencias de la Computación e Inteligencia Artificial.

Gracias también al Dr. Hilario Ramirez, nuestro biólogo de cabecera, por hacerse tiempo para atendernos y por su ayuda en el problema de emparejamiento estructural de moléculas.

Quiero expresar mi gratitud general hacia profesores del exterior (J. Szustakowsky, J. Beasley, A. Lokketangen, N. Krasnogor, P. Moscato, etc) que amablemente me enviaron copias de sus trabajos y a todos aquellos que colocan los resultados de sus investigaciones en Internet para que otros podamos acceder a ellos.

Fuera del ámbito académico, quiero agradecer a Sergio y Natalia (y flia.) por su amistad y por hacer que la estadía en Granada sea más agradable. Siempre recordaré las idas a la playa, las fiestas de fin de año y muchas y buenas cenas compartidas.

A Juan José, Leticia, Erandi y Luis, los "mexicanos", por compartir con nosotros sus experiencias que en el fondo también son las nuestras.

A Ariel, Karin y Adrián, simplemente por estar dispuestos a ayudar en lo que haga falta.

A mis compañeras de despacho en el Mecenas, Eva y Belén, por hacer de mi lugar de trabajo en los últimos meses un sitio agradable y acogedor.

También quiero agradecer a mi amigo Natalio Krasnogor, con quien en 1996 empezamos el Grupo Biocom siendo de los primeros grupos en Argentina en dedicarnos al área de Bioinformática. Sin su insistencia para que realizara la preinscripción al doctorado en 1998, vaya uno a saber que estaría haciendo en estos momentos. Su disponibilidad para leer varios manuscritos horribles y sus constantes sugerencias, además del trabajo en conjunto que seguimos realizando, hacen que lo considere un colega excepcional. Quizás con un poco de suerte, el futuro nos encuentre trabajando juntos y en el mismo lugar. Si no, Internet y el teléfono nos ayudarán.

Al otro lado del Atlántico están mis padres, Beba y Héctor, junto con mi hermana Vero y Aquiles, y mi sobrino Valentín a quien al momento de escribir estas líneas solo conozco por fotos. Ellos saben que a pesar de la distancia,

siempre los tengo presentes y que aunque las circunstancias hacen que estemos un poco lejos, siempre aspiro a que podamos volver a estar un poco más cerca. Espero poder dar a mi hija el mismo amor que ellos me dieron y que ha hecho de mí la persona que soy.

Este último año ha sido especialmente duro para todos los argentinos. Desde afuera, la angustia y la impotencia se multiplican y ojalá que entre todos seamos capaces de generar las alternativas que permitan reconstruir un país que esta vez sea justo, solidario y para todos.

Finalmente, me faltan las palabras para agradecerle a Gaby todo su esfuerzo, comprensión, apoyo, ayuda y amor para que esta aventura que empezamos juntos en el 98 llegara a buen puerto. Sin ella nada de esto hubiera sido posible y espero que los proximos años sean tan buenos como estos últimos.

Desde hace ahora 6 meses, tenemos un sol propio en nuestro universo particular: Martina. Ella nos llena de vida cada día con su sonrisa, sus progresos constantes y su alegría. Es el centro de nuestra atención y la "responsable" de que esta tesis no estuviera terminada antes.

Así que Gaby y Martina, mis dos amores, esta tesis que resume gran parte del trabajo de casi 4 años, va dedicada a ustedes.

Introducción

Las técnicas de resolución de problemas basadas en computadoras son elementos claves en la sociedad de la información, y hoy en día es practicamente imposible concebir cualquier proceso de toma de decisiones que no cuente con la ayuda de programas y algoritmos. La complejidad de la realidad actual nos obliga a enfrentarnos a una clase amplia de problemas para los cuales no se disponen herramientas para resolverlos exactamente utilizando una cantidad moderada de recursos.

Uno de los campos donde este tipo de problemas complejos aparece en gran número, es el de la Bioinformática, la cual entendemos como un área en la frontera entre la Biología y las Ciencias de la Computación cuyo principal objetivo es el desarrollo y uso de técnicas matemáticas y computacionales para ayudar en la resolución de problemas de la biología molecular.

Un prerequisito para tratar con esta clase de problemas, es poseer la habilidad de dominar un arsenal de técnicas o algoritmos que hayan sido desarrollados para una variedad de condiciones. Naturalmente, sería deseable que para cada problema pudieramos encontrar la solución óptima a través de algoritmos exactos pero, lamentablemente, las modelizaciones asociadas a gran parte de los problemas a resolver en Bioinformática, dan lugar a problemas NP-Completos y por lo tanto, deben ser abordados mediante técnicas heurísticas. A pesar de ello, esta área resulta especialmente atractiva puesto que la resolución de estos problemas y la obtención de nuevos mecanismos de solución tienen y tendrán impacto en, principalmente, dos áreas: las Ciencias de la Computación y la Biología.

Tanto en Bioinformática como en otros campos, sería deseable disponer de

herramientas computacionales que ayuden a obtener soluciones iniciales "razonablemente buenas" para problemas nuevos, y que sean lo suficientemente flexibles para permitir la incorporación del conocimiento específico del problema que se vaya generando. La idea subyacente a este proceso es que al enfrentarnos con un problema nuevo, en ocasiones debemos intentar solucionarlo (o proporcionar alguna solución satisfactoria) en forma rápida y con mínimo conocimiento. Luego, se inicia un proceso iterativo que tiene las siguientes fases: 1)*aprender del problema*, 2) *incorporar conocimiento al algoritmo*, 3) *obtener nueva solución* y 4) *reiniciar el ciclo*. Por aprendizaje del problema entendemos, la incorporación de restricciones, la detección de posibles cotas, la reformulación del problema original, la búsqueda de problemas similares ya resueltos, etc.

Entre los métodos heurísticos que cumplen el requisito anterior podemos citar los conocidos genericamente como *métodos de búsqueda por entornos* o de *búsqueda local* cuyas principales ventajas son su simplicidad, flexibilidad y generalidad. Bajo el concepto básico de búsqueda local existen una amplia variedad de algoritmos que difieren esencialmente en muy pocos aspectos.

Por otro lado, la posibilidad de disponer de un conjunto de heurísticas para resolver un problema determinado, implica disponer en última instancia, de un conjunto de posibles soluciones. Es decir, mediante n algoritmos, se obtiene un conjunto $S = \{s_1, s_2, \ldots, s_n\}$ de n soluciones potencialmente diferentes. Por lo tanto, también sería deseable disponer de una especie de molde o "framework" de heurísticas que pueda ser adaptado a diferentes problemas y que permita capturar las ideas esenciales de un conjunto de métodos disponibles. De esta forma, mediante un único algoritmo adaptable A, potencialmente podríamos obtener el mismo conjunto de soluciones S.

No debemos olvidar que la utilización de una heurística nos obliga a plantear la resolución de problemas en términos de satisfacción de un usuario, que en el caso de Bioinformática, puede ser un biólogo. En ocasiones, será suficiente con que el método retorne la solución de mejor costo que pueda encontrar, pero en ocasiones se deben proveer soluciones que no solo sean "buenas" en términos de costo, sino que también posean características adicionales que pue-

den llegar ser de naturaleza subjetiva y por lo tanto, modelizables mediante conjuntos difusos.

Por lo tanto, dadas la necesidad de desarrollar herramientas que sirvan para obtener soluciones a problemas de la Bioinformática, la importancia que tienen las heurísticas como mecanismos de resolución de problemas, y la utilidad que presentan los elementos básicos de la lógica difusa para modelizar conceptos vagos o abstractos, en este trabajo proponemos combinar un método heurístico clásico de optimización combinatoria con elementos básicos de la teoría de conjuntos difusos para dar lugar a una herramienta de optimización robusta, aplicable en Bioinformática, que cumple los dos aspectos deseables planteados anteriormente: 1) que ayude a obtener soluciones iniciales "razonablemente buenas" para problemas nuevos, siendo flexible para permitir la incorporación del conocimiento específico del problema; 2) que funcione como un "framework" de heurísticas, capturando bajo un mismo esquema el comportamiento de varios métodos.

Nuestro Método de Búsqueda por Entornos, Adaptativo y Difuso, llamado *FANS* por su nombre en inglés (Fuzzy Adaptive Neighborhood Search) es basicamente un método de búsqueda local, denominado Difuso porque las soluciones se evalúan en términos de valoraciones difusas y Adaptativo, porque su comportamiento varía en función del estado de la búsqueda. Veremos como a través del componente difuso, el método es capaz de capturar el comportamiento cualitativo de otras heurísticas basadas en búsqueda por entornos, con lo cual, será posible obtener un "framework" de heurísticas.

Es bien sabido que los métodos de búsqueda local presentan el inconveniente de quedar atrapados en óptimos locales. En la bibliografía se suelen presentar tres variantes clásicas para potenciar el esquema básico de mejoramiento iterativo. La primera consiste en reiniciar el método desde soluciones iniciales diferentes dando lugar a los métodos multi arranque; la segunda consiste en aceptar transiciones hacia soluciones que empeoren el costo y finalmente, la tercera variante consiste en incorporar memoria al algoritmo para evitar ciclos y dirigir la búsqueda hacia regiones no exploradas del espacio. Sin embargo, casi todos estos algoritmos se basan en la utilización de un único operador de

movimiento (induciendo un vecindario determinado) para generar soluciones, sin tener en cuenta que la optimalidad local de una solución solo vale bajo cierta definición particular de vecindario.

Es por eso que otro elemento distinguido de nuestra propuesta, es la utilización de varios vecindarios en la etapa de búsqueda local, lo cual permite diseñar estrategias de búsqueda mas robustas y representa además, un mecanismo para escapar de optimos locales.

Por lo tanto, y con el objetivo de mostrar la aplicación de *FANS* a problemas de la Bioinformática, se plantea primero la presentación y el análisis experimental de *FANS* en varios dominios, para mostrar su utilidad como herramienta de optimización general. Para acometer estas tareas y alcanzar los objetivos propuestos, esta memoria se encuentra organizada en 4 capítulos, cuyos contenidos se describen a continuación.

El Cap. 1 está dedicado a introducir los elementos esenciales de la disciplina de la optimización combinatoria y las técnicas heurísticas. Se describen propiedades de las llamadas *Metaheurísticas* y se incluyen algunos ejemplos de las mismas: recocido simulado, búsqueda tabú, búsqueda local iterativa, búsqueda por entornos variable y algoritmos genéticos. Para cada una de ellas existe una bibliografía muy amplia cuyo relevamiento no está dentro de los objetivos de este trabajo; por lo tanto, en cada caso se describen las definiciones básicas de cada técnica y se proveen referencias que pueden servir de guía para el lector interesado.

En el Cap. 2 se presenta nuestra propuesta: *FANS*. En primer lugar se describen los elementos básicos de *FANS* para luego llegar al esquema general del algoritmo. A continuación se describe detalladamente la llamada valoración difusa, poniendo el acento en la motivación y utilidad de esta componente, y destacando especialmente como puede ser utilizado para variar el comportamiento del algoritmo, dando lugar a un “framework” de métodos de búsqueda por entornos. Luego se describen una serie de tareas orientadas a verificar la utilidad del segundo elemento novedoso de *FANS*: el uso de varios operadores en el proceso de búsqueda. Para ello se presenta primero una justificación basada en la idea de “landscapes” y luego se muestran experimentos y resultados

que dan soporte experimental a dicha justificación.

Siendo *FANS* un método heurístico, es natural analizar su rendimiento o "performance" sobre un conjunto de problemas y realizar comparaciones frente a otros métodos. Estas tareas son las abordadas en el Cap. 3 donde bajo las hipótesis de mínimo conocimiento del problema y cantidad limitada de recursos, se realizan experimentos comparativos sobre problemas de la mochila simple y con múltiples restricciones y sobre problemas de minimización de funciones reales. Se describe en detalle cada experimento realizado y se plantean algunos aspectos tenidos en cuenta para que las comparaciones sean justas. Además se desarrollan experimentos complementarios para analizar la calidad de las soluciones investigadas por *FANS*, y para evaluar la influencia que tiene en los resultados la utilización de diferentes definiciones para el llamado "administrador de vecindario".

Una vez analizada la utilidad de *FANS* como herramienta de optimización, llegamos al Cap. 4 donde abordamos la aplicación de *FANS* a problemas de Bioinformática. Como primer paso, se presentan algunas nociones muy elementales de biología, se describe brevemente el proyecto genoma humano y luego se analiza qué entendemos por Bioinformática.

Como veremos, la gama de problemas que abarca esta área es muy amplia y por lo tanto es imposible abarcarlos a todos. Es por eso que en esta memoria nos centraremos solamente en dos de ellos que tienen especial importancia: el problema de predicción de estructura en modelos basados en retículos, y el problema de emparejamiento estructural de moléculas. Ambos problemas se definen adecuadamente y se revisan algunos trabajos previos.

Posteriormente, se muestra como se puede aplicar *FANS* en cada uno de ellos. Para el primer problema, se utiliza nuestro algoritmo para evaluar dos aspectos: primero, para analizar la influencia que tiene la codificación de las soluciones en los resultados obtenidos; y segundo, para comparar *FANS* frente a un algoritmo genético y verificar una hipótesis respecto a la posibilidad de evitar el uso de una población de soluciones.

En el segundo problema, se analiza como influyen dos factores en el comportamiento de *FANS*: el tamaño del patrón a buscar (una de las entradas del

problema) y el valor de un parámetro del método que se utiliza para controlar la aceptación de soluciones.

Finalmente, se resumen las tareas realizadas en esta memoria, se presentan las conclusiones obtenidas y se establecen nuevas líneas de investigación, algunas de las cuales ya se encuentran en desarrollo.

Capítulo 1

Optimización Combinatoria y Metaheurísticas

Debido a la importancia práctica de los problemas de optimización combinatoria, se han desarrollado una amplia variedad de algoritmos para intentar resolverlos. Estos algoritmos pueden clasificarse como *exactos* o *aproximados*. Mientras los primeros garantizan la obtención del óptimo de cualquier instancia finita del problema en un tiempo acotado, los segundos sacrifican optimalidad, poniendo énfasis en la obtención de soluciones satisfactorias en tiempo reducido. En vista que un gran número de problemas combinatorios resultan NP-Completos[1], la utilización de algoritmos aproximados es y será un área de intensa actividad.

Dentro de los algoritmos aproximados se suelen distinguir los basados en métodos constructivos (partiendo de una solución "vacía" se van agregando componentes hasta obtener una solución completa) y los basados en búsqueda local (comenzando desde una solución dada, se la va reemplazando por soluciones mejores pertenecientes a un vecindario predefinido). En los ultimos años, estos esquemas o métodos básicos de resolución se han incluído en esquemas más generales que reciben el nombre genérico de *Metaheurísticas*.

En este capítulo de caracter introductorio, se presentarán una serie de

[1] Para una revisión de los aspectos de teoría de la computación y la complejidad se sugiere consultar [42]. En [31] se puede encontrar una recopilación de problemas NP-Completos

elementos bien conocidos con el objetivo de unificar los términos que se utilizarán a lo largo de la memoria. En primer lugar se introducirán algunas definiciones básicas relativas a la optimización combinatoria y posteriormente se describirán las características que en general poseen las metaheurísticas. Finalmente se presentarán algunos ejemplos de metaheurísticas exitosas. Dado que para cada una de ellas existe una bibliografía muy amplia cuyo relevamiento no está dentro de los objetivos de este trabajo; además de describir las características esenciales de las mismas, se proveen referencias que pueden servir de guía para el lector interesado.

1.1 Definiciones Preliminares

Una instancia de un problema de optimización combinatoria P se define como $(\mathcal{S}, f)$ y está compuesta por

- un conjunto de variables $X = \{x_1, \ldots, x_n\}$
- dominios de las variables $D_1, \ldots, D_n$
- un conjunto de restricciones entre variables
- una función objetivo $f : D_1 \times \ldots \times D_n \rightarrow R^+$

El conjunto de todas las asignaciones factibles es $\mathcal{S} = \{s = (x_1, v_1), .., (x_n, v_n)\}$ con $v_i \in D_i$, $i = 1, \ldots, n$ y tal que s satisface las restricciones. $\mathcal{S}$ recibe el nombre de espacio de búsqueda.

Resolver el problema implica encontrar la solución que minimice/maximice el valor de la función objetivo. Es decir, asumiendo un problema de minimización, encontrar un $s^* \in \mathcal{S}$ tal que $f(s^*) \leq f(w)$ para todo $w \in \mathcal{S}$. La solución s^* se denomina el óptimo de $(\mathcal{S}, f)$.

Ahora, sea $(\mathcal{S}, f)$ una instancia de un problema de optimización. Se define la función o estructura de vecindario $\mathcal{N}$ como una aplicación $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$, el cual determina para cada solución $s \in \mathcal{S}$ el conjunto $\mathcal{N}(s) \subseteq \mathcal{S}$ de soluciones "cercanas" (en algún sentido) a s. El conjunto $\mathcal{N}(s)$ se denomina el vecindario

de la solución s. Ejemplos de vecindarios surgen naturalmente por ejemplo, si se utiliza una función de distancia:

$$dist : \mathcal{S} \times \mathcal{S} \rightarrow R$$

Se puede definir el vecindario de una solución s como:

$$\mathcal{N}(s) = \{y \in \mathcal{S} \mid dist(s, y) \leq \epsilon\}$$

También se podría utilizar la siguiente definición:

$$N(s) = \{y \in \mathcal{S} \mid \rho(y, s) = True\}$$

donde $\rho(a, b)$ es algún predicado booleano.

En general, las soluciones y, se obtienen a partir de la aplicación de un operador $\mathcal{O}$, al que se suele denominar "Move" o "Pivote", que modifica en algún sentido la solución actual s para obtener nuevas soluciones. Este operador suele incluir algún procedimiento de aleatorización, con lo cual sucesivas aplicaciones de $\mathcal{O}(s)$ permiten obtener soluciones y diferentes.

Tiene sentido entonces hablar del vecindario de s bajo $\mathcal{O}$ y definirlo como

$$\mathcal{N}_{\mathcal{O}}(s) \ = \{\hat{s}_i \mid \hat{s}_i = \mathcal{O}_i(s)\}$$

donde $\mathcal{O}_i(s)$ representa la i-esima aplicación de $\mathcal{O}$ sobre s.

Una vez definido de alguna manera apropiada el vecindario de una solución s como $\mathcal{N}(s)$, y dada una solución inicial s_0, es posible definir un esquema de mejoramiento iterativo que comience buscando alguna solución $s_1 \in N(s_0)$ tal que s_1 verifique alguna propiedad. Por ejemplo, que mejore el costo asociado.

Posteriormente, s_1 pasa a ser la solución actual y el proceso se repite hasta que se satisfaga algún criterio de parada o no sea posible obtener ninguna solución $s_{k+1} \in N(s_k)$ que satisfaga los requerimientos establecidos. En este momento, se establece que s_k es un óptimo local.

Esta clase de métodos reciben también el nombre de *búsqueda local*[1] (*BL* de ahora en adelante) y en la Fig. 1.1(a) se muestra su pseudocódigo elemental. La rutina *mejorar(x)* devuelve, si es posible, una nueva solución y del vecindario tal que y sea mejor que s. En caso contrario, retorna "NO" y

se devuelve la solución actual, la cual se corresponde con un mínimo/máximo local. Este esquema básico recibe el nombre de Mejoramiento Iterativo o HillClimbing.

Existen, al menos, dos estrategias básicas para implementar la rutina *mejorar(x)*: la estrategia *First*, que retorna la primera solución del vecindario que mejore el costo, y la estrategia *Best*, que explora el vecindario de forma exhaustiva y retorna la solución con mejor costo. Ambas estrategias, tambien llamadas *reglas de pivot*, finalizan cuando se alcanza un óptimo local y por lo tanto la calidad final de la solución está fuertemente influenciada por las definiciones de $\mathcal{S}$, f y $\mathcal{N}$.

A pesar de su simplicidad, estos métodos presentan un inconveniente importante: suelen quedar atrapados en mínimos o máximos locales. Como consecuencia es necesario extender los métodos mediante mecanismos adicionales que permitan hacer frente a esta situación.

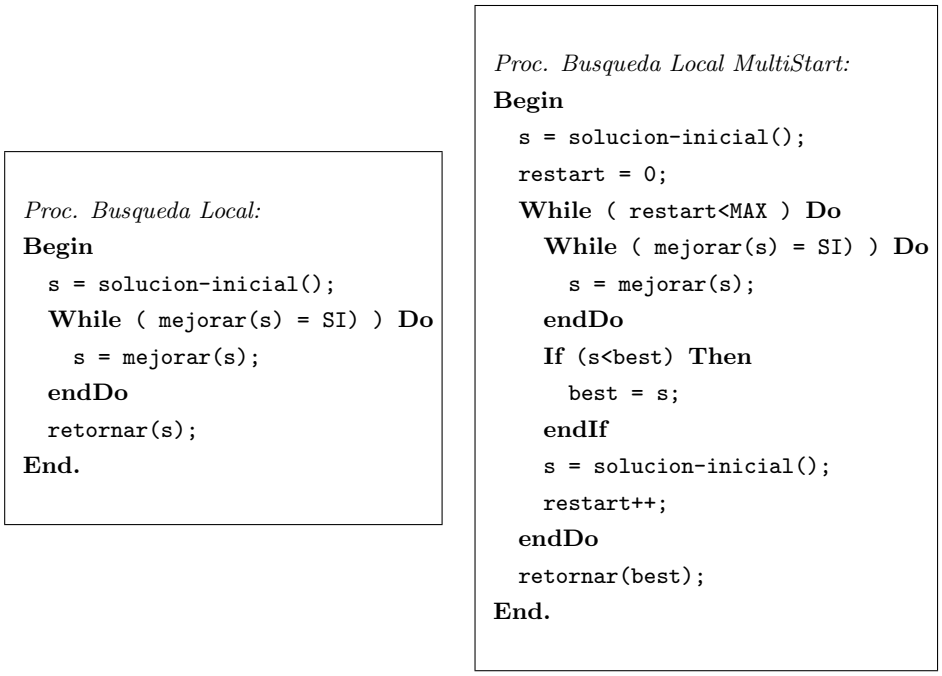
La forma más simple de extender el esquema se presenta en la Fig.1.1(b), donde simplemente se reinicia la búsqueda desde una nueva solución, cuando la actual ya no puede ser mejorada.

Otro mecanismo para evitar el problema de los optimos locales, consiste en incluir el esquema básico de búsqueda local en esquemas de un nivel superior, dando lugar a las denominadas metaheurísticas. Entre ellas, vale la pena citar la Búsqueda Tabú [48], los algoritmos evolutivos [9], el recocido simulado [64], la búsqueda local iterativa [84] y la Búsqueda por Entornos Variable (VNS, Variable Neighborhood Search) [54, 55].

1.2 Propiedades de las Metaheurísticas

El término *metaheurística* deriva de la composición de dos palabras griegas. *Heurística* proviene del verbo *heuriskein* que significa "encontrar", mientras que el sufijo *meta* significa "mas allá" o "de nivel superior". Aunque no existe una definición formal de que es y no es una metaheurística, las dos siguientes propuestas representan adecuadamente la noción general del término:

Definición 1 [93]:"una metaheurística se define formalmente como un proceso



Proc. Busqueda Local:
Begin
  s = solucion-inicial();
  While (mejorar(s) = SI)) **Do**
    s = mejorar(s);
  endDo
  retornar(s);
End.

Proc. Busqueda Local MultiStart:
Begin
  s = solucion-inicial();
  restart = 0;
  While (restart<MAX) **Do**
    While (mejorar(s) = SI)) **Do**
      s = mejorar(s);
    endDo
    If (s<best) **Then**
      best = s;
    endIf
    s = solucion-inicial();
    restart++;
  endDo
  retornar(best);
End.

(a) (b)

Figura 1.1: *Esquemas básicos de Búsqueda Local*

iterativo que guia una heurística subordinada, combinando de forma inteligente diferentes conceptos para explorar y explotar el espacio de búsqueda."

Definición 2 [117]:"una metaheurística es un proceso maestro iterativo que guia y modifica las operaciones de heuristicas subordinadas para producir, de forma eficiente, soluciones de alta calidad. En cada iteración, puede manipular una solución (completa o incompleta) o un conjunto de soluciones. Las heurísticas subordinadas pueden ser procedimientos de alto o bajo nivel, o simplemente una búsqueda local o método constructivo"

Además, una metaheurística debe poseer todas o algunas de las siguientes propiedades [19, 94]:

- *Simplicidad*: debe basarse en conceptos claros que permitan una aplicación general.
- *Eficiencia*: debe proveer resultados razonables para un gran número

de instancias realistas de un problema. Idealmente, para los conjuntos estandar disponibles debería alcanzar los mejores valores disponibles.

- *Efectividad*: deben utilizar tiempo y recursos computacionales moderados para obtener soluciones aceptables.
- *Robustez*: el rendimiento debe ser consistente sobre un amplio rango de instancias y no solamente buenas en un conjunto de pruebas y regulares en cualquier otro caso.
- *"Amigable"*:deben estar bien definidas, ser simples de entender y utilizar. Por ej. incluir un número reducido de parámetros.

Dada la amplia variedad de metaheurísticas existentes, se han intentado realizar varias clasificaciones de las mismas. Por ejemplo, en función del origen del método, se habla de algoritmos "bioinspirados" (algoritmos genéticos, de colonia de hormigas, etc) vs. "no bioinspirados". Otra posibilidad es categorizarlos en función de la utilización o no de memoria, o en función del uso de una función objetivo estática o dinámica. Una clasificación interesante es la que surge al diferenciar los métodos que mantienen una única solución, frente a los que mantienen un conjunto o población de soluciones [28]. En la sección siguiente se presentan brevemente algunos ejemplos de metaheurísticas pertenecientes a la primer clase y en el mismo estilo, se describen los algoritmos genéticos como ejemplo más claro de algoritmo basado en un conjunto de soluciones.

1.3 Ejemplos de Metaheurísticas

El campo de la optimización mediante heurísticas es muy amplio, con un gran número de científicos trabajando en él, y es un área que genera una cantidad de publicaciones y resultados dificilmente comparables con los de cualquier otra área. Esto hace que existan una variedad de métodos, y variantes de los mismos,cuya descripción por su extensión, está fuera de los objetivos de esta memoria. Nótese que sobre mediados de mayo, el buscador *www.google.com* utilizando como palabras clave el nombre en inglés de 10 técnicas heurísticas

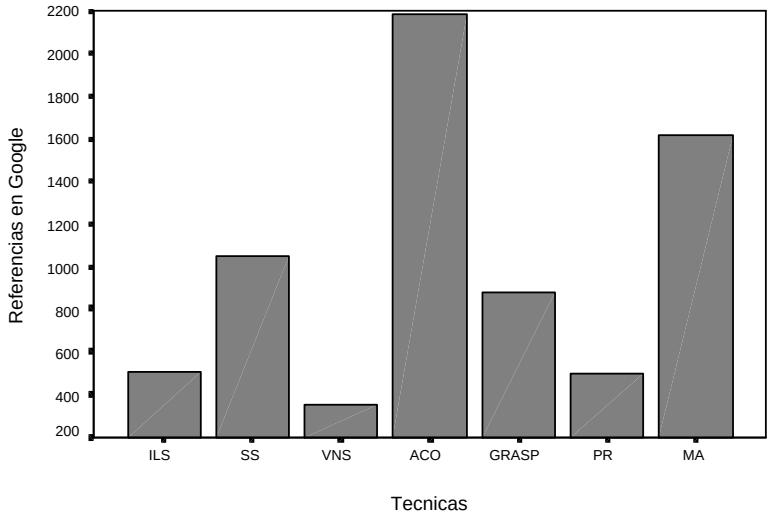


Figura 1.2: *Número de referencias obtenidas con Google para cada técnica. Las búsquedas se realizaron a mediados de mayo utilizando solamente el nombre en inglés del método.*

conocidas, devolvió la cantidad de referencias mostradas en la Fig. 1.2 donde las abreviaturas utilizadas son *ILS: Iterated Local Search, SS: scatter search, VNS: Variable Neighborhood Search, ACO: Ant Colony Optimization, GRASP: Greedy Randomized Adaptive Search Procedure, PR: Path Relinking, MA: Memetic Algorithms*.

El gráfico no incluye algoritmos genéticos (con 153000 referencias), recocido simulado (66800), ni búsqueda tabú (16200) para no desvirtuar la escala. Se puede observar claramente que para ninguna de las búsquedas existen menos de 350 referencias. Naturalmente esto no indica que todas ellas sean útiles[2], pero si sirven como evidencia del dinamismo del tema.

A continuación describiremos solamente un conjunto de metaheurísticas que presentan una serie de características que suelen estar presentes en casi todos los demás métodos (aceptación de soluciones peores, utilización de memoria, rearranque desde soluciones especiales, etc). El lector interesado en analizar otros métodos puede referirse al trabajo de Pardalos y Resende [94],

[2]Incluso algunos no dudarían en tildar la situación como de mera charlatenería

Corne et.at [28] o Back et.al [7] para compilaciones recientes sobre el área de las metaheuristícas. Para finales de 2002 se espera la aparición del trabajo de Glover y Kochenberger [47].

Recocido Simulado

El método de Recocido Simulado o "Simulated Annealing" (*SA*, de ahora en adelante), desarrollado por Kirpatrick en 1983 [71] está inspirado en el proceso físico de enfriamiento utilizado en el tratamiento de metales y pertenece a una clase más amplia de algoritmos conocidos como *Treshold Algorithms* o algoritmos de umbral.

El esquema básico de un algoritmo de umbral se presenta en la Fig. 1.3. En cada iteración el algoritmo selecciona una solución vecina de la actual y compara la diferencia entre los costos. Si esta diferencia es menor que cierto umbral, la solución vecina pasa a ser la solución actual y el proceso se repite. La secuencia $(t_k,\ k = 0, 1, 2, \cdots)$ denota los umbrales y se cumple que $t_k = c_k, k = 0, 1, 2, \cdots$, donde $c_k \geq 0, c_k \geq c_{k+1}$ y $lim_{k\to\infty} c_k = 0$. El uso de umbrales positivos provoca que soluciones mucho peores sean aceptadas en forma limitada. A medida que el algoritmo progresa, los valores de umbral son reducidos hasta alcanzar 0, momento en el cual unicamente las soluciones que mejoren el costo son aceptadas.

En el recocido simulado, el valor de t_k es una variable aleatoria que sigue cierta función de probabilidad. *SA* utiliza umbrales aleatorios con valores entre 0 e infinito y la aceptación de soluciones peores que la actual esta gobernada por el siguiente criterio:

$$rand(0, 1) < exp^{(f(x_{i+1})-f(x_i))/T} \tag{1.1}$$

donde el valor de T representa un parámetro que recibe el nombre de "temperatura" y $rand(0, 1)$ es un número aleatorio entre 0 y 1 con distribución uniforme.

La estrategia de *SA* es comenzar con una temperatura inicial "alta", lo cual proporciona una probabilidad también alta de aceptar movimientos de empeoramiento. En cada iteración se reduce la temperatura y por lo tanto

```
Procedure Algoritmo de Umbral:
Begin
  s_a = solucion-inicial();
  While ( no-finalizacion ) Do
    s_v = GenerarVecino(s_a);
    If (f(s_v − s_a) < t_k) Then
      s_a = s_v;
    endIf
    k=k+1;
  endDo
End.
```

Figura 1.3: *Esquema de un Algoritmo de Umbral*

las probabilidades de aceptar soluciones peores también disminuyen. De este modo, se comienza la búsqueda con una etapa de exploración, a la que continúa una etapa de explotación, a medida que la ejecución avanza.

Se considera que el esquema de enfriamiento es uno de los elementos cruciales en la efectividad de *SA* y bajo ciertas condiciones, está probado que *SA* converge al óptimo global con probabilidad tan cercana a 1 como se desee. Desafortunadamente, en la práctica dichas condiciones son imposibles de alcanzar.

En la literatura especializada, se pueden encontrar numerosas aplicaciones exitosas de *SA* en problemas de optimización combinatoria. Por ejemplo, en [107, 116, 106] se incluye *SA* como una técnica moderna de optimización y se describen algunas aplicaciones. Ejemplos de utilización de *SA* en problemas de "scheduling" se muestran en [23, 114] y en [64] se presentan aspectos que contrastan las cuestiones teóricas frente a las prácticas.

Búsqueda Tabú

La Búsqueda Tabú (*BT*, de ahora en adelante) presentada por Glover a principios de los 90, [45, 46], incorpora una lista de movimientos prohibidos (tabú),

para forzar la búsqueda en regiones diferentes del espacio y evitar problemas de ciclos.

La BT comienza desde una solución inicial s_a, construye un entorno $\mathcal{N}(s_a)$ y selecciona la mejor solución $s_b \in N(s)$. De la misma manera que un movimiento m permitió transformar $s_a \rightarrow s_b$, existirá el movimiento inverso m^{-1} para trasnformar $s_b \rightarrow s_a$. Con el objetivo de evitar ciclos, el movimiento m^{-1} se agrega a la lista tabú. El procedimiento continúa desde s_b hasta que cierta condición de finalización se verifique.

Los movimientos permanecen como tabú sólo durante un cierto número de iteraciones, aunque hay excepciones. Cuando un movimiento tabú proporciona una solución mejor que cualquier otra previamente encontrada, su clasificación tabú puede eliminarse. La condición que permite dicha eliminación se denomina *criterio de aspiración*.

Las restricciones tabú y el criterio de aspiración de la BT juegan un papel dual en la restricción y guía del proceso de búsqueda. Las restricciones tabú, permiten que un movimiento sea admisible si no está clasificado como tabú, mientras que si el criterio de aspiración se satisface, permite que un movimiento sea admisible aunque este clasificado como tabú.

La longitud de la lista tabú es un parámetro. Su definición no es trivial ya que si es demasiado pequeño pueden ocurrir ciclos, mientras que si es demasiado grande, puede restringir en exceso la búsqueda. Esta lista recibe también el nombre de memoria de corto plazo. En ocasiones, y como base para las estrategias de exploración/explotación, también se utilizan memorias de término intermedio y largo plazo.

La BT incorpora varios conceptos más que los comentados aquí, como por ejemplo el de oscilación estratégica. El lector interesado puede consultar [48] para una revisión reciente de todos los aspectos teóricos y prácticos relacionados con la BT, o acceder a `www.upt.pt/tabusearch` para encontrar informes sobre implementaciones modernas del método. Finalmente, es necesario destacar que en el año 2000, se obtuvo una prueba de convergencia de BT [53].

```
Procedure Busqueda Tabu:
Begin
  s_a = solucion-inicial();
  ListaTabu = {};
  While ( no-finalizacion ) Do
    s_b = Mejorar(s_a, N(s_a), ListaTabu);
    s_a = s_b;
    Actualizar(ListaTabu);
  endDo
  retornar(s);
End.
```

Figura 1.4: *Esquema de la Búsqueda Tabu*

Búsqueda Local Iterativa

La Búsqueda Local Iterativa (Iterated Local Search, ILS de ahora en adelante) [84] es una metaheurística simple y de propósito general que aplica iterativamente un procedimiento de búsqueda local sobre modificaciones de la solución actual.

La idea básica del método es realizar una trayectoria que pase unicamente a través del conjunto de óptimos locales $\mathcal{S}^*$ en lugar de utilizar todo el espacio $\mathcal{S}$. Naturalmente no es posible construir una estructura de vecindario en $\mathcal{S}^*$. Sin embargo, la trayectoria $s_1^*, s_2^*, \ldots, s_t^*$ puede obtenerse sin tener tal estructura explícitamente de la forma mostrada en la Fig. 1.5.

Se pueden observar tres elementos que deben ser definidos: un procedimiento **Perturbar**, que perturba la solución actual s_a dando lugar a una solución intermedia s_p; un procedimiento **BusquedaLocal** que transforma s_p en un óptimo local s_{ol}; y finalmente un criterio **Aceptar?()** para decidir desde que punto se inicia la próxima iteración.

Cualquier procedimiento de búsqueda local puede instanciarse en **BusquedaLocal** y en general, la operación de perturbación debe utilizar un tipo de movimiento diferente al utilizado en **BusquedaLocal**. Además **Perturbar**,

```
Proc. Busqueda Local Iterativa:
Begin
  /* s_a:  solucion actual */
  /* s_p:  solucion intermedia o perturbada */
  /* s_ol:  solucion localmente optima */
  /* H: historia o memoria */
  s_1 = solucion-inicial();
  s_a = BusquedaLocal(s_1);
  Repeat Until ( finalizacion ) Do
    s_p = Perturbar(s_a,H);
    s_ol = BusquedaLocal(s_p);
    s_a = Aceptar?(s_a,s_ol,H);
  endDo
End.
```

Figura 1.5: *PseudoCódigo de la Búsqueda Local Iterativa*

tambien llamado *Kick Move*, debe ser suficientemente fuerte para permitir escapar del mínimo local, pero también lo suficientemente débil como para mantener algunas características de la solución actual.

Búsqueda por Entornos Variable

La metaheurística de Búsqueda por Entornos Variable (Variable Neighborhood Search, VNS de ahora en adelante), propuesta por Hansen y Mladenovic en [54] está basada en la observación que los mínimos locales tienden a agruparse en una o varias zonas del espacio de búsqueda.

Por lo tanto una vez que se consigue un óptimo local, resulta beneficioso aprovechar la información que este contiene. Por ejemplo, que varias variables puedan tener valores iguales o cercanos al del óptimo global. VNS propone explorar vecindarios cercanos, y progresivamente más lejanos de la solución actual en la búsqueda de mejores soluciones.

El esquema básico del algoritmo se presenta en la Fig. 1.6. Asociada con cada iteración de VNS, existe una solución actual s_a y un vecindario de orden

```
Proc. Busqueda por Entornos Variable:
Begin
  /* N_k, k = 1,...,k_max, estructuras de vecindarios */
  /* s_a: solucion actual */
  /* s_p: solucion vecina de s_a */
  /* s_ol: solucion localmente optima */
  Repeat Until ( finalizacion ) Do
    k=1;
    Repeat Until ( k = k_max ) Do
      /* generar vecino s_p del k-esimo vecindario de s_a (s_p ∈ N_k(s_a)) */
      s_p = ObtenerVecino(s_a,N_k);
      s_ol = BusquedaLocal(s_p);
      If (s_ol es mejor que s_a) Then
        s_a = s_ol;
      Else
        k=k+1;
      endIf
    endDo
  endDo
End.
```

Figura 1.6: *PseudoCódigo de la Búsqueda por Entornos Variable*

k. En cada iteración se ejecutan dos pasos: primero, se genera una solución vecina de s_a, $s_p \in \mathcal{N}_k(s_a)$; segundo, se aplica un procedimiento de búsqueda local sobre s_p, dando lugar a una nueva solución s_{ol}. Si s_{ol} mejora la solución actual s_a, entonces la búsqueda se reinicia desde s_{ol} utilizando $k = 1$. En caso contrario, se hace $k = k + 1$ y se repite el proceso desde s_a. El algoritmo para después de cierto número de veces que se haya realizado la exploración de la secuencia completa $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{k_{max}}$

Una variante interesante pero poco explorada de VNS consiste en la utilización de diferentes esquemas de búsqueda local o de varios operadores en la fase de mejoramiento iterativo denominada *variable neighborhood descent*.

Algoritmos Genéticos

Los algoritmos genéticos (AG, de ahora en adelante) son una familia de modelos computacionales inspirados en los mecanismos de herencia y evolución natural (*supervivencia del más apto*).

A diferencia de los métodos comentados, estas técnicas mantienen un conjunto de potenciales soluciones, tienen algún proceso de selección basado en la calidad de las mismas, e incorporan operadores que permiten modificar o generar nuevas soluciones.

Como se sabe, en los organismos biológicos la información hereditaria es pasada a través de los cromosomas. Varios organismos se agrupan formando una población, y aquellos que mejor se adaptan son los que más probabilidades tienen de sobrevivir y reproducirse. Algunos de los supervivientes son seleccionados por la naturaleza para ser cruzados y así producir una nueva generación de organismos. Esporádicamente, los genes de un cromosoma pueden sufrir ligeros cambios (mutaciones).

Mediante una imitación de este mecanismo, los algoritmos genéticos exploran el espacio de soluciones asociado a un determinado problema. En la Fig. 1.7 se presenta el esquema básico de un AG. En cada iteración t, se mantiene una *población* de *individuos* $P(\mathrm{t}) = \{x_1^t, \ldots, x_n^t\}$ donde cada individuo representa una potencial solución del problema a resolver. Cada solución x_i^t se evalúa para obtener cierta medida de su calidad o *fitness*. Luego, se genera una nueva población $P(t+1)$ tomando como base a los mejores individuos; algunos de los cuales sufren transformaciones a partir de la aplicación de operadores "genéticos". Típicamente, estos operadores son: *cruzamiento* o *crossover* (nuevos individuos son generados a partir de la combinación de dos o más individuos), y *mutación* (nuevos individuos son generados a partir de pequeños cambios en un individuo).

La idea intuitiva del operador de cruce es permitir el intercambio de información entre individuos de la población. La mutación, en cambio, permite incorporar nueva información que no se puede generar a partir de la clausura transitiva del cruce sobre la población.

Podemos entender el mecanismo de selección si planteamos cada iteración

```
Procedure Algoritmo Genetico:
Begin
  t = 0;
  inicializar(P(t));
  evaluar(P(t));
  While ( no-finalizacion ) Do
    t = t + 1;
    Seleccionar P(t) desde P(t-1);
    Hacer-Cruzamientos(P(t));
    Aplicar-Mutaciones(P(t));
  endDo
  retornar mejor solucion;
End.
```

Figura 1.7: *Esquema básico de un Algoritmo Genético*

del AG como un proceso en dos etapas. En la primera, se aplica algún mecanismo de selección sobre la "población actual" para crear una "población intermedia"; en la segunda etapa, a partir de esta población, se aplica cruce y mutación para generar una "nueva población".

El resultado del proceso de selección es un *conjunto de entrecruzamiento* o *mating pool* formado por los mejores individuos de la población. En base a ellos, la etapa de cruce elegirá los "padres" y generará los "hijos"; sobre la salida de esta etapa, se ejecutará la etapa de mutación. La nueva población puede estar formada solamente por los hijos, o por padres e hijos en alguna proporción adecuada.

En el tema de AG's existe una amplia diversidad de enfoques, variantes y elementos, lo cual provoca que cualquier intento de resumen sea forzosamente incompleto. El lector interesado en profundizar en estos temas, puede consultar [90, 7, 33] y las referencias allí citadas.

Capítulo 2

FANS: una Heurística para Problemas de Optimización

El diseño, la construcción y la búsqueda de algoritmos que permitan resolver problemas que surgen en la vida real, son objetivos esenciales de las Ciencias de la Computación. A pesar de la alta complejidad que suelen presentar dichos problemas, deben ser resueltos pues son de notable importancia. Ambos aspectos, dificultad e importancia, han potenciado el desarrollo de técnicas heurísticas que, aunque pueden llevar a la obtención de soluciones sub óptimas, son capaces de resolver el problema en términos de "satisfacción" del usuario. En general, los métodos heurísticos proveen soluciones que resultan "buenas" en términos de cierta función de costo, pero también es posible que, además, el usuario considere otras características que en ocasiones pueden ser de naturaleza subjetiva y por lo tanto, modelizables mediante conjuntos difusos.

Desde nuestro punto de vista, en las últimas décadas se produjo un flujo de información importante desde áreas clásicas, como la Investigación Operativa o la Teoría de Control, hacia el área de los Conjuntos y Sistemas Difusos y que permitió obtener resultados verdaderamente fructíferos. Hoy en día, términos como "Control Difuso", "Programación Matemática Difusa", "Sistemas basados en reglas difusas", etc, resultan familiares a cualquier investigador. Para ver ejemplos de esta interacción, el lector interesado puede referirse a [63, 32] y también a la recopilación bibliográfica de Cordón et.al [27], que a pesar de

abarcar hasta 1996, presenta una buena muestra de la combinación de conjuntos y lógica difusa con algoritmos genéticos.

Sin embargo, esta interacción no ha sido usual en el sentido contrario. En este capítulo mostraremos como un método clásico de optimización combinatoria se puede combinar con elementos básicos del área de los conjuntos y sistemas difusos para dar lugar a una herramienta de optimización adaptativa y robusta. Para ello, se plantea como objetivo global la presentación de las ideas esenciales de un nuevo método de búsqueda por entornos, adaptativo y difuso, denominado *FANS* por las iniciales de su nombre en inglés: *Fuzzy Adaptive Neighborhood Search*. *FANS*, incorpora como elementos destacables la utilización de una valoración difusa o subjetiva para evaluar las soluciones, y el uso de varios operadores para explorar el espacio de búsqueda.

La motivación para la utilización de la valoración subjetiva, parte de la idea que en ocasiones se suele hablar de soluciones *Diferentes*, *Similares*, *Razonables*, etc, siendo estas características de naturaleza subjetiva o vaga. Por lo general, las reglas de transición entre soluciones de los métodos reseñados son de naturaleza crisp, es decir, consideran a una solución como "mejor" o no, "diferente" o no, etc. En el contexto de *FANS*, las soluciones son vistas como elementos de un conjunto difuso[1] cuya función de pertenencia es dicha valoración difusa, la cual representa cierta propiedad denominada genericamente P. De esta forma, las soluciones tienen asignado un grado de pertenencia a dicho conjunto (por ejemplo, el conjunto de soluciones $P = aceptables$) lo cual permite definir subconjuntos de soluciones del tipo *muy* P, *algo* P, *no* P, etc. A través de esta valoración difusa, el comportamiento de *FANS* puede ser modificado de forma tal que se logra reflejar el comportamiento cualitativo de otros métodos de búsqueda local. De esta manera, veremos que *FANS* es en realidad un "framework" o molde de métodos de búsqueda local.

La utilización de varios operadores en el proceso de búsqueda es un área relativamente poco explorada, pero con una justificación muy simple: que una solución sea óptima localmente bajo cierto vecindario (inducido por un operador), no implica que dicha condición de optimalidad se verifique en otro

[1]Los elementos básicos de conjuntos difusos y variables lingüisticas no serán descritos aquí. El lector interesado puede referirse a [122, 29, 121].

vecindario[2]. Esta idea de variación del vecindario a explorar, se utiliza en una variante muy poco investigada de VNS denominada *variable neighborhood descent* [55].

Siendo además un método de búsqueda por entornos, resulta simple de comprender, implementar y mejorar, y por lo tanto resulta atractivo como herramienta para realizar las primeras aproximaciones para la resolución de un problema dado.

En este capítulo se detallan las tareas realizadas para alcanzar el objetivo descripto, utilizando la siguiente estructura: en la Sección 2.1 se describen los elementos básicos de *FANS* y se presenta el esquema general del algoritmo. Posteriormente en la Sección 2.2 se describe la motivación y utilidad de la valoración difusa y se muestra como este componente puede ser utilizado para variar el comportamiento del algoritmo, dando lugar a un "framework" de métodos de búsqueda por entornos. En la Sección 2.3 se presentan brevemente algunos detalles de implementación y se citan otros trabajos relacionados. Finalmente, la Sección 2.4 esta dedicada a verificar la utilidad del segundo elemento novedoso de *FANS*: el uso de varios operadores en el proceso de búsqueda. Para ello se presenta primero una justificación basada en la idea de "landscapes" y luego se muestran experimentos y resultados que dan soporte experimental a dicha justificación. Finalmente, se presentan las conclusiones en la sección 2.5.

2.1 Presentación de *FANS*

FANS se define como un método de búsqueda por entornos, adaptativo y difuso. Es un método de búsqueda por entornos porque el algoritmo realiza transiciones desde una solución de referencia a otra de su entorno, produciendo "trayectorias" o caminos. Es adaptativo porque su comportamiento varía en función del estado de la búsqueda y finalmente es considerado difuso porque las soluciones son evaluadas,además de la función objetivo, mediante una *valoración difusa* que representa algún concepto subjetivo o abstracto.

[2]Salvo que esta solución sea el óptimo global.

FANS está basado en cuatro componentes principales: un operador, para construir soluciones; una valoración difusa, para cualificarlas; un administrador de operación, para adaptar el comportamiento o características del operador; y un administrador de vecindario, para generar y seleccionar una nueva solución.

En esta sección se describen con más detalle los elementos principales del algoritmo para lo cual se utilizarán las siguientes convenciones: $s_i \in \mathcal{S}$ es una solución del espacio de búsqueda, $\mathcal{O}_i \in \mathcal{M}$ es un operador de Modificación o de Movimiento del espacio de operadores; $\mathcal{P}$ representa el espacio de parámetros (tuplas de valores) y $\mathcal{F}$ representa el espacio de los conjuntos difusos cuyos elementos se denotan como $\mu_i()$.

Con estas definiciones, *FANS* queda especificado con la 7-tupla siguiente:

$$Fans(\mathcal{NS}, \mathcal{O}, \mathcal{OS}, \mu(), Pars, (cond, accion))$$

donde $\mathcal{NS}$ es el *Administrador de Vecindario*, $\mathcal{O}$ es el operador utilizado para construir soluciones, $\mathcal{OS}$ es el *Administrador de Operación* y $\mu()$ es una valoración difusa o subjetiva.

Además, $Pars$ representa un conjunto de parámetros y el par $(cond, accion)$ es una regla de tipo $IF cond THEN accion$ que se utiliza para detectar y actuar en consecuencia cuando la búsqueda se ha estancado.
A continuación se describen las características principales de cada componente y posteriormente se presenta el esquema del algoritmo.

2.1.1 El Operador de Modificación

Dada una solución de referencia $s \in \mathcal{S}$, el operador de modificación $\mathcal{O}_i \in \mathcal{M}$, con $\mathcal{O} : \mathcal{S} \times \mathcal{P} \rightarrow \mathcal{S}$, construye nuevas soluciones s_i a partir de s.

Cada aplicación del operador sobre la misma solución s debe devolver una solución diferente. Es decir, debe existir algún elemento de aleatoriedad en su definición. Un requerimiento adicional es que el operador provea algún elemento o parámetro adaptable $t \in \mathcal{P}$ para controlar su comportamiento. Por lo tanto, utilizaremos $\mathcal{O}^t$ para referirnos al operador con ciertos parámetros t.

2.1.2 La Valoración Difusa

Como planteamos en la introducción del capítulo, en el contexto de *FANS* las soluciones son evaluadas (además de la función objetivo) en términos de la *valoración difusa*. La motivación de este aspecto parte de la idea que en ocasiones se suele hablar de soluciones *Diferentes, Similares, Razonables*, etc, siendo estas características de naturaleza subjetiva o vaga. Una forma de modelizar adecuadamente este tipo de conceptos vagos se logra representando la valoración difusa mediante un conjunto difuso $\mu() \in \mathcal{F}, \text{con } \mu : \mathcal{R} \rightarrow [0,1]$. Como sinónimo de valoración difusa, son aceptables términos como *concepto difuso* o *propiedad difusa*, lo que permite hablar de soluciones que verifican dicha propiedad en cierto grado.

Por lo tanto, la valoración difusa nos permite obtener el grado de pertenencia de la solución al conjunto difuso representado por $\mu()$. Por ejemplo, teniendo el conjunto difuso de las soluciones "buenas", consideraremos el grado de bondad de la solución de interés. Así, dadas dos soluciones $a, b \in \mathcal{S}$ podemos pensar en cuan *Similar* es a con b, o cuan *Cerca* están, o también cuan *Diferente* es b de a. *Similar, Cerca, Diferente* serán conjuntos difusos representados por funciones de pertenencia $\mu()$ apropiadas y naturales en el área de la optimización combinatoria.

2.1.3 El Administrador de Operación

Como ya se planteó, *FANS* utiliza varios operadores de modificación en el proceso de búsqueda y este Administrador de Operación $\mathcal{OS}$ es el responsable de definir el esquema de adaptación u orden de aplicación de los diferentes operadores utilizados.

El cambio de operador, o lo que es lo mismo, la ejecución del administrador, puede realizarse en cada iteración, o cuando el estado de la búsqueda así lo requiera. Esta adaptación puede estar basada en estadísticas del algoritmo tales como cantidad de evaluaciones de la función de costo realizadas, o teniendo en cuenta medidas particulares del proceso de búsqueda, como por ejemplo el número de iteraciones sin cambios en la mejor solución hallada, etc.

Basicamente se sugieren dos posibilidades para el Administrador. En la

primera, que utilizamos en este trabajo, el administrador $\mathcal{OS}$ ajustará los parámetros del operador y en consecuencia, devolverá un operador cuyo comportamiento será diferente: $\mathcal{OS}\ (\mathcal{O}^{ti}) \Rightarrow \mathcal{O}^{tj}$.

La segunda alternativa, consiste en disponer de una familia de operadores de modificación $\{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_k\}$. Bajo esta situación, $\mathcal{OS}$ podría definir el orden de aplicación de los mismos. Dado que cada operador implica un vecindario diferente, obtendremos una estructura que recuerda a la de *variable neighborhood search* (VNS) [54]. Sin embargo, nuestro esquema es diferente al de VNS ya que *FANS* no utiliza un método explícito de búsqueda local ni requiere una métrica de distancia entre soluciones como la necesaria en VNS.

Claramente el operador y el administrador de operación estarán fuertemente acoplados y es factible pensar en una situación donde operadores específicos requieran administradores o gestores específicos.

2.1.4 El Administrador de Vecindario

Este componente es el responsable de generar y seleccionar una nueva solución del vecindario. Podemos verlo como una función cuyo tipo es:

$$\mathcal{NS} : \mathcal{S} \times \mathcal{F} \times \mathcal{M} \times \mathcal{P} \Rightarrow \mathcal{S}$$

En el contexto de *FANS* se utilizan dos tipos de vecindarios: el *operativo* y el *semantico*, ambos definidos respecto a cierta solución de referencia s.

Dados el operador $\mathcal{O}$ y la solución actual s, se define el vecindario operativo como:

$$\mathcal{N}(s) = \{\hat{s}_i \mid \hat{s}_i = \mathcal{O}_i(s)\} \tag{2.1}$$

donde $\mathcal{O}_i(s)$ indica la i-ésima aplicación de $\mathcal{O}$ sobre s.

Para la definición del *vecindario semantico* de s se utiliza la valoración difusa $\mu()$, dando lugar a la siguiente definición:

$$\hat{\mathcal{N}}(s) = \{\hat{s}_i \in \mathcal{N}(s) \mid \mu(\hat{s}_i) \geq \lambda\} \tag{2.2}$$

Es decir, $\hat{\mathcal{N}}(s)$ representa el λ-corte del conjunto difuso de soluciones representado por $\mu()$. En otras palabras, las soluciones de interés serán aquellas que satisfagan nuestra valoración con, al menos, cierto grado λ.

La operación del administrador es simple: primero se ejecuta un *generador* para obtener soluciones del vecindario semántico a partir de varias aplicaciones del operador de modificación $\mathcal{O}$. Posteriormente, el procedimiento *selector* debe decidir cuál de estas soluciones retornar teniendo en cuenta: los grados de pertenencia de dichas soluciones, su costo o una combinación de ambos valores.

Por ejemplo, si estuvieramos utilizando una valoración difusa de "Similaridad" con respecto a la solución actual, el selector podría utilizar reglas de selección como las siguientes:

- *Mejor:* Devolver la solución más similar disponible,
- *Peor:* Devolver la menos similar,
- *Primera:* Devolver la primer solución suficientemente similar.

Naturalmente, tambien podría utilizarse el costo de esas soluciones similares para obtener reglas de selección como:

- *MaxMax:* De las soluciones similares, retornar la de mayor costo,
- *MaxMin:* De las soluciones similares, retornar la de menor costo.

2.1.5 Parámetros Globales

FANS mantiene un conjunto de parámetros globales $Pars \in \mathcal{P}$ para llevar el registro del estado de la búsqueda y este conjunto es utilizado por varios componentes para decidir las acciones a tomar.

2.1.6 Manejo de optimos locales

Es sabido que todos los métodos de búsqueda local presentan como principal inconveniente, el quedar atrapados en óptimos locales. *FANS* maneja esta situación a través de dos mecanismos que se utilizan para escapar de dichos óptimos.

El primero mecanismo está contenido en la variación del operador en las etapas de búsqueda y se describirá en la sección 2.4. El otro mecanismo de escape esta basado en el par $(cond, accion)$ donde *cond*, llamada

$HayEstancamiento?() : \mathcal{P} \rightarrow [True, False]$, es utilizada para determinar cuando hay suficiente evidencia de que la búsqueda esta definitivamente estancada. Cuando *cond* se verifique, entonces se ejecutará la acción $accion = Escape()$. Por ejemplo, se podría reiniciar el algoritmo desde una nueva solución inicial, reiniciar desde una solución obtenida a partir de una modificación especial de la actual, o cualquier otra opción que se considere adecuada.

Figura 2.1: *Esquema de* FANS

Procedimiento FANS*:*

```
Begin
  /* O : el operador de modificacion */
  /* μ() : la valoracion difusa */
  /* S_cur, S_new : soluciones */
  /* NS: el administrador de vecindario */
  /* OS: el administrador de operacion */
  Inicializar Variables();
  While ( not-fin ) Do
    /* ejecutar el administrador de vecindario NS */
    S_new = NS(O, μ(), S_cur);
    If (S_new es ''buena'' en terminos de μ()) Then
      S_cur := S_new;
      adaptar-Valoracion-Difusa(μ(), S_cur);
    Else
      /* No fue posible encontrar una solucion buena con  */
      /* el operador actual */
      /* Sera modificado */
      O := OS-ModificarOperador(O);
    endIf
    If (HayEstancamiento?()) Then
      Escape();
    endIf
  endDo
End.
```

2.1.7 El Algoritmo

En esta sección se describe el esquema de *FANS*, el cual se muestra en la Fig. 2.1. Se puede observar que cada iteración comienza con una llamada al administrador de vecindario $\mathcal{NS}$ con los siguientes parámetros: la solución actual S_{cur}, la valoración difusa $\mu()$ y el operador de modificación $\mathcal{O}$. Como resultado de la ejecución de $\mathcal{NS}$ pueden ocurrir 2 cosas: se pudo encontrar una solución vecina aceptable S_{new} (en términos de $\mu()$), o no se pudo.

En el primer caso, S_{new} pasa a ser la solución actual y los parámetros de $\mu()$ son adaptados. Por ejemplo, si $\mu()$ representa la similaridad respecto a la solución actual, entonces la valoración difusa debe ser adaptada para reflejar este cambio en la solución de referencia. Si $\mathcal{NS}$ no pudo retornar una solución aceptable, es decir, en el vecindario inducido por el operador no se pudo encontrar una solución suficientemente buena, entonces se aplica el nuevo mecanismo de escape: se ejecuta el administrador de operación OS el cual retornara una versión modificada del operador $\mathcal{O}$. La próxima vez que $\mathcal{NS}$ se ejecute, dispondrá de un operador modificado para buscar soluciones, y por lo tanto es posible que el resultado sea diferente.

La condición $HayEstancamiento?()$ será verdadera cuando (por ejemplo) se hayan realizado $Tope$ llamadas a OS sin haber obtenido mejoras en la solución actual. Es decir, se probaron varios operadores y no se obtuvieron mejoras. En este caso, se ejecutará el procedimiento $Escape()$, se evaluará el costo de la nueva solución y se adaptará la valoración difusa $\mu()$. Posteriormente, se reinicia la búsqueda.

Debe quedar claro que lo que varía en cada iteración son los parámetros utilizados en las llamadas a $\mathcal{NS}$. El algoritmo comienza con $\mathcal{NS}$ $(s_0, \mathcal{O}^{t_0}, \mu_0)$. Si $\mathcal{NS}$ puede retornar una solución aceptable, entonces en la siguiente iteración la llamada será $\mathcal{NS}$ $(s_1, \mathcal{O}^{t_0}, \mu_1)$; es decir cambia la solución actual y por lo tanto se modifica la valoración difusa. Si en cierta iteración l, $\mathcal{NS}$ no puede retornar una solución aceptable, entonces se ejecutará el administrador de operación el cual devolverá una versión modificada del operador. La iteración siguiente, $\mathcal{NS}$ será llamado con $\mathcal{NS}$ $(s_l, \mathcal{O}^{t_1}, \mu_l)$.

Durante la ejecución del algoritmo pueden ocurrir dos situaciones pro-

blemáticas: primero, se realizaron varias llamadas $\mathcal{NS}(s_j, \mathcal{O}^{t_i}, \mu_j)$ con $i \in [1, 2, \ldots, k]$, lo cual significa que se probaron k formas diferentes de obtener una solución aceptable y ninguna tuvo éxito; o segundo , la búsqueda se está moviendo entre soluciones aceptables pero en las ultimas m llamadas no se consiguió mejorar la mejor solución de todas las visitadas hasta el momento.

Cuando cualquiera de las dos situaciones ocurra, se ejecutará el procedimiento $Escape()$ lo cual llevará a alguna de las dos posibilidades siguientes: $\mathcal{NS}$ $(\hat{s}_0, \mathcal{O}^{t_j}, \hat{\mu}_0)$ o $\mathcal{NS}$ $(\hat{s}_0, \mathcal{O}^{t_0}, \hat{\mu}_0)$, donde $\hat{s}_0$ es una nueva solucion inicial (generada aleatoriamente, por ejemplo), y $\hat{\mu}_0$ es la valoración difusa obtenida en función de $\hat{s}_0$. Respecto al operador, se puede mantener con los mismos parámetros ($\mathcal{O}^{t_j}$) o también se lo puede "reinicializar" ($\mathcal{O}^{t_0}$).

El criterio de parada del algoritmo se basa en la verificación de una condición externa. Por ejemplo, el algoritmo finaliza cuando el número de evaluaciones de la función de costo alcanza cierto límite o cuando se realizaron un número predeterminado de evaluaciones.

2.2 La Valoración Difusa como Mecanismo de Variación del Comportamiento de *FANS*

El comportamiento global de *FANS* depende de la definición de sus componentes y de la interacción entre ellos. A través de la utilización de diferentes definiciones y parámetros para la valoración difusa se consigue que el algoritmo se comporte de diferente manera, lo que da lugar, en consecuencia, a variaciones en los resultados que se puedan obtener.

Antes de seguir adelante, es necesario proveer una definición para el administrador de vecindario. Por simplicidad, utilizaremos un esquema denominado $First$ que funciona de la siguiente manera: dada una solución s, y un operador $\mathcal{O}$, el procedimiento *generador* del Administrador obtendrá soluciones del vecindario operativo una valoración, las cuales serán analizadas por el *selector* utilizando la valoración difusa $\mu()$, y un nivel mínimo de calidad requerido λ. La regla de selección empleada simplemente establece que se devolverá la primera solución analizada $\hat{s}$ que verifique $\mu(s, \hat{s}) \geq \lambda$. A lo sumo

se utilizarán cierto número de intentos y este valor debe especificarse en cada caso.

La valoración difusa representará cierta noción de "Aceptabilidad" y asumiremos un problema de minimización.

Teniendo en mente el esquema de *FANS* presentado anteriormente en la Fig. 2.1, discutiremos en esta sección como el algoritmo puede ser adaptado para reflejar el comportamiento cualitativo de otros métodos tradicionales de búsqueda local [17].

Comportamiento tipo Caminos Aleatorios

Para obtener este comportamiento, todo lo que se necesita es considerar cualquier solución del vecindario como aceptable. Por lo tanto, utilizando una valoración difusa con $\mu(f(s), f(\hat{s})) = 1 \ \forall \ \hat{s} \in \mathcal{N}(s)$, cualquier solución del vecindario operacional tendrá la opción de ser seleccionada. De esta manera, *FANS* se comportará como un método que produce caminos aleatorios.

Comportamiento tipo Hill Climbing

En este caso, para obtener el comportamiento deseado es necesario utilizar una valoración difusa tal que $\mu(s, \hat{s}) = 1$ if $f(\hat{s}) < f(s)$ y asignar $\lambda = 1$. De esta manera, unicamente serán consideradas como aceptables aquellas soluciones que mejoren el costo actual. El método de Hill Climbing termina cuando se alcanza un óptimo local. En el contexto de *FANS*, la obtención de un método multiarranque es directo.

Comportamiento tipo Recocido Simulado

En el método de Recocido Simulado, la aceptación de soluciones esta gobernada por un parámetro externo denominado "Temperatura". Las soluciones que mejoran el costo actual siempre son aceptadas. Aquellas que lo empeoran también pueden ser aceptadas con cierta probabilidad, que será alta al inicio de la ejecución. A medida que la búsqueda progresa, la temperatura disminuye decrementando entonces la probabilidad de aceptación de soluciones peores. Hacia el final de la ejecución, solo las soluciones mejores son aceptadas.

Para reflejar este comportamiento mediante la valoración difusa, se puede utilizar la siguiente definición de "Acceptabilidad":

$$\mu(q,s) = \begin{cases} 0.0 & \text{si } f(q) > \beta \\ \beta - f(q)/\beta - f(s) & \text{si } f(s) \leq f(q) \leq \beta \\ 1.0 & \text{si } f(q) < f(s) \end{cases}$$

donde f es la función objetivo, s es la solución actual y q es una solución del vecindario operativo.

El valor β representa el límite para lo que se considera aceptable. Este valor es clave ya que en última instancia determina que soluciones pertenecen o no al vecindario semántico. Si se define β como una función $h(s,q,t)$ donde t sea un parámetro que represente por ejemplo el número actual de iteraciones, el límite de deterioro aceptable puede ser reducido a medida que la búsqueda progresa. Hacia el final de la ejecución, solo aquellas soluciones mejores que la actual serán tenidas en cuenta. Un ejemplo para la función h es:

$$h = f(s) * (1 + \frac{1}{1 + e^{(f(s)-f(q))/t}}) \tag{2.3}$$

donde el segundo término de la suma representa la distribución de probabilidad de Boltzmann.

En términos más generales, β representan el umbral de aceptabilidad y por lo tanto, su variación esta reflejando una clase de algoritmos más amplia, los algoritmos de umbral.

Comportamiento tipo Búsqueda Tabú

El esquema básico de BT utiliza una historia o memoria H para condicionar y guiar la búsqueda. Tipicamente, esta memoria se utiliza para restringir o descartar la generación de ciertas soluciones del vecindario.

Naturalmente dicha estructura de memoria no puede ser capturada mediante la valoración difusa. Sin embargo, si en el contexto de *FANS* se deseara incorporar memoria, esta debería incluirse dentro del procedimiento *Generador* del administrador de vecindario, el cual tiene la responsabilidad de generar las soluciones.

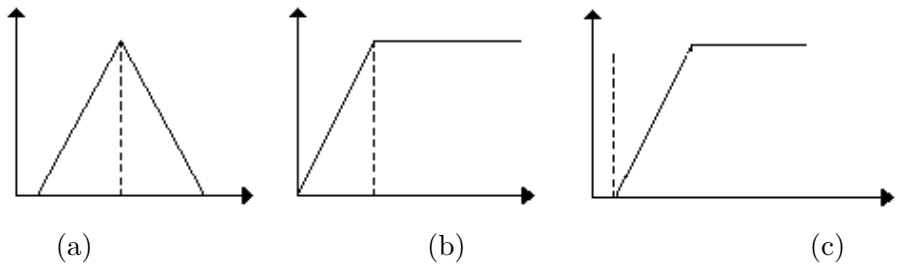


Figura 2.2: *Ejemplos de Valoraciones Difusas.*

La BT retorna siempre la mejor solución disponible. En este caso, es necesario utilizar un administrador diferente de $First$ que permita obtener un conjunto de soluciones del cual el procedimiento *selector* retorne la "mejor" en términos de la valoración difusa.

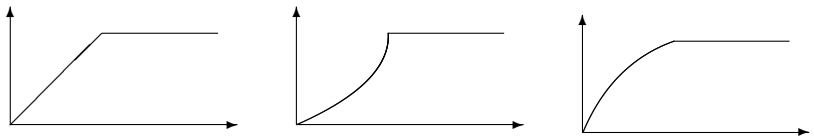
FANS como heurística de propósito general

En los párrafos anteriores mostramos como *FANS* puede ser ajustado para reflejar el comportamiento cualitativo de otras técnicas, principalmente a través de la manipulación de la valoración difusa. En consecuencia, y como se había planteado, podemos decir que *FANS* representa un "framework" para métodos simples de búsqueda local.

Sin embargo, *FANS* puede plantearse como una heurística en sí misma capaz de mostrar una variedad de comportamientos muy amplia. Este potencial queda claro si consideramos que la valoración difusa representa el criterio de un decisor buscando soluciones para un problema.

Por ejemplo, en la Figura 2.2 se representan varios criterios posibles para aceptabilidad. Las definiciones están basadas en el costo de cierta solución actual, representado en los gráficos por una línea de puntos.

La definición triangular (a), representa un decisor buscando soluciones similares en costo pero diferentes en estructura. Este tipo de comportamiento puede aparecer cuando las soluciones presentan características dificiles de cuantificar (por ejemplo, aspectos estéticos) con lo cual el decisor desea obtener un conjunto de opciones de costo similar para luego hacer su elección en términos de otros criterios diferentes al costo. La definicion central (b) repre-



(a) (b) (c)

Figura 2.3: *Aplicación de los modificadores lingüisticos Algo, en (b), y Muy, en (c), sobre la definición de Aceptabilidad (a).*

senta un decisor "insatisfecho" con la solución actual. Cualquier opción que la mejore, tendrá un valor máximo de aceptabilidad. Finalmente la tercera opción (c), representa un decisor "conservador" en el sentido que para cambiar la solución actual, la mejora obtenida debe ser considerable.

Otra posibilidad de conseguir comportamientos diferentes, se obtiene mediante la aplicación de modificadores lingüisticos sobre cierta definición dada de una valoración difusa. Un ejemplo aparece en la Figura 2.3 donde se muestra en primer lugar la definición de Aceptabilidad y a continuación, las definiciones respectivas para *Algo Aceptable* y *Muy Aceptable*.

En síntesis, *FANS* brinda la posibilidad de variar su comportamiento y en última instancia las soluciones que se pueden obtener, de forma simple y comprensible. Si consideramos que cada comportamiento permite obtener resultados cuantitativamente diferentes, y que para cada problema resulta mejor un algoritmo que otro [3], la potencialidad de *FANS* como herramienta genérica de optimización resulta clara.

2.3 Comentarios Sobre la Implementación

En la sección anterior se describió como *FANS*, además de capturar el comportamiento cualitativo de otros métodos de búsqueda local, es en sí misma una heurística de propósito general. En este sentido, podemos considerar a *FANS* como un "framework" de métodos de búsqueda local.

FANS también puede considerarse un framework desde el punto de vista de la metodología de diseño e implementación utilizada, la cual se realizó con

[3] Aunque a priori es imposible saberlo

un lenguaje orientado a objetos (C++). Un framework es una clase especial de librería de software que consiste en una jerarquía de clases abstractas. Los frameworks se caracterizan por el mecanismo de *control inverso* para la comunicación con el código del usuario: las funciones del framework "llaman" a las definidas por el usuario y no al revés, como ocurre con las librerías o bibliotecas estandar.

El framework provee las partes invariantes, el esquema genérico, del algoritmo mientras que el usuario debe proveer los detalles específicos del problema, evitando así, la necesidad de codificar desde cero todo el algoritmo. De esta manera, la división en componentes de *FANS*, junto con la posibilidad de derivar y componer clases (consecuencia de la utilización de un lenguaje orientado a objetos), permiten al usuario la implementación de diferentes versiones de *FANS* y el testeo de nuevas ideas en forma relativamente simple y clara.

En la actualidad pueden encontrarse frameworks para el desarrollo de algoritmos, como $EasyLocal++$ [43], o el trabajo de Andreatta et.al. [4] destinados a la implementación de algoritmos de búsqueda local; $HotFrame$ [39] que incorpora diferentes metaheuristicas o $MAFRA$ [78], orientado a los algoritmos meméticos.

El principal objetivo de estos sistemas es simplificar la tarea de los usuarios que necesiten implementar algorimos heurísticos. En muchos casos, se plantea que el principal inconveniente que poseen es el relativo a que la parte de implementación asociada al problema a tratar (la cual es responsabilidad del usuario) domina el costo total de la implementación. Por lo tanto se cuestiona cual es la utilidad de proveer un molde para la parte genérica del algoritmo cuando es la más simple de implementar.

De acuerdo con [43], existen dos respuestas a esta cuestión. La primera se refiere al convencimiento general que la resolución de problemas complejos se dirige hacia la combinación de métodos y técnicas cuya interacción incrementa la dificultad de la parte, teoricamente, "fácil" de los algoritmos. Este incremento aparece tanto en complejidad como en costos de programación. En segundo lugar, los beneficios de la utilización de estos sistemas no se refieren solamente a la cantidad de código que un usuario deba escribir, sino también

a conceptos de modularidad y claridad conceptual. La utilización de frameworks fuerza al usuario a ubicar cada porción de código en el lugar "correcto" facilitando los procedimientos de extensión y depuración.

2.4 Utilización de Múltiples Operadores en el Proceso de Búsqueda

Para concluir la presentación de *FANS*, en esta sección se analiza el segundo elemento novedoso que incorpora el método: la utilización de varios operadores en el proceso de búsqueda lo cual sirve a dos propósitos: primero, como mecanismo para potenciar la búsqueda local, y segundo como mecanismo de escape de optimos locales.

Tomando como base la idea de *"un operador, un landscape"*, introducida por T. Jones en [67, 68], se presentará una justificación sobre el efecto que produce la utilización de varios operadores en el espacio de búsqueda.

Posteriormente, se realizarán una serie de experimentos sobre un conjunto de instancias del problema de la mochila, para demostrar que, aún bajo un esquema de optimización muy simple, la utilización de varios operadores produce errores menores que cuando se dispone de un único operador.

2.4.1 "Un Operador, un Landscape"

Es usual intentar imaginar o visualizar como un algoritmo se "mueve" a través de un espacio tridimensional o *"landscape"*[4], como el mostrado en la Fig 2.4, pasando por "picos", "valles", o dando "saltos" hacia regiones más prometedoras del espacio de búsqueda [67].

En general, se plantea que un pico es un punto cuyo costo es mejor que el de todos los puntos de alrededor, pero dicha definición de "pico" solo tiene sentido en el contexto de un vecindario y casi nunca este hecho es tenido en cuenta. Por ejemplo, cuando se utiliza representación binaria, es usual considerar el hipercubo como el landscape a explorar. Así, en el contexto de los algoritmos

[4]Dado que no existe una traducción adecuada para este término en este contexto, se seguirá utilizando en inglés

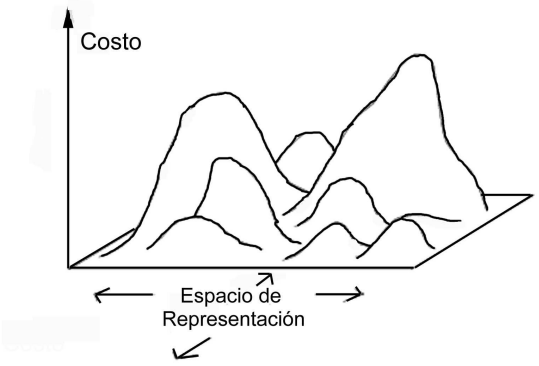


Figura 2.4: *Una descripción gráfica del concepto general de "landscape".*

evolutivos, se suele plantear que el operador de cruce produce "saltos" largos en el landscape, aún cuando el cruce no induzca el hipercubo.

En [67], se plantea que cada operador se "mueve" en su propio landscape y se propone la siguiente formalización: un landscape $\mathcal{L}$ puede ser visto como un grafo dirigido $G_{\mathcal{L}} = (V, E)$. Un operador θ puede considerarse como un evento estocástico que ocurre en cierto contexto $v \in V$ y cuya salida es cierta variable aleatoria W con alguna función de distribución de probabilidad. La probabilidad del evento $W = w$ para un $w \in V$ en el contexto de v se denota como $\theta(v, w)$. El vecindario θ de v, se define como $N_\theta(v) = \{w \in V : \theta(v, w) > 0\}$. Si ocurre que $\theta(v, w) > 0$, entonces $G_{\mathcal{L}}$ contendrá una arista desde v a w. Los vertices en $G_{\mathcal{L}}$ se corresponden con las posibles entradas y salidas de los operadores y no existen restricciones para considerar que los vertices puedan corresponder con "multisets".

Bajo este esquema, por ejemplo, los AG's utilizan 3 landscapes diferentes, cada uno asociado con el operador de cruce, el de mutación y el de selección respectivamente. La Fig.2.5 muestra este funcionamiento donde el AG aparece realizando movimientos entre soluciones en el landscape de mutación. Luego, se seleccionan pares de individuos que formarán los vértices en el landscape de cruce. Se aplica el operador y luego toda la población se transforma en un vértice del landscape de selección. Finalmente, la población se descompone en individuos que formarán nuevamente los vértices en el landscape de mutación.

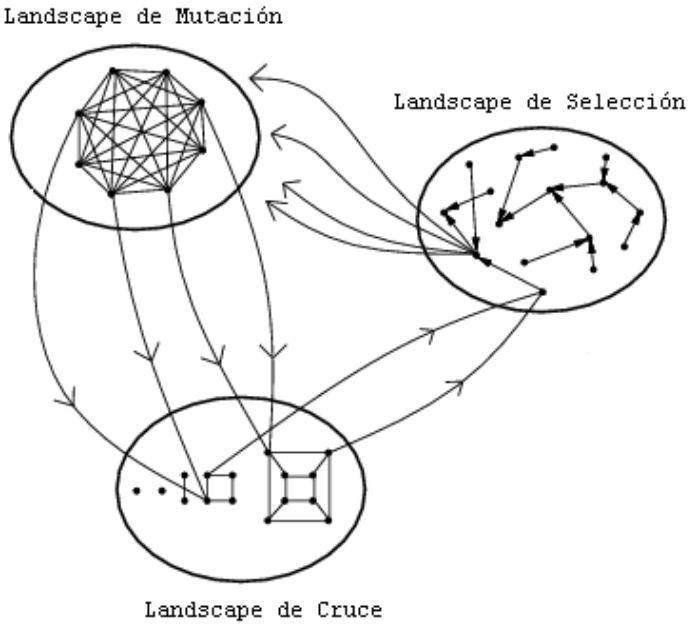


Figura 2.5: *Una visión simplificada de la operación de un* AG *sobre tres landscapes*

De esta manera y generalizando a otros métodos, podemos plantear que un algoritmo simplemente se mueve a través de las aristas del landscape inducido por cada operador que utilice.

En el contexto de *FANS*, si bien la definición del vecindario operativo es fija, la idea de cambiar el operador de modificación $\mathcal{O}$ es inducir un cambio en las soluciones que se pueden obtener a partir de cierta solución de referencia x dada. Es decir, estaremos cambiando el landscape a recorrer.

Si pensamos en un problema con soluciones binarias, una solución actual x, un operador $\mathcal{A}$ que modifique k variables, y un operador $\mathcal{B}$ que modifique $k+1$ variables, entonces resulta que $\mathcal{N}_{\mathcal{A}}(x) \neq \mathcal{N}_{\mathcal{B}}(x)$. Un ejemplo trivial es el siguiente: supongamos que $x = 0000$, $\mathcal{A}$ modifica una variable y $\mathcal{B}$ modifica dos, entonces los vecindarios inducidos son:

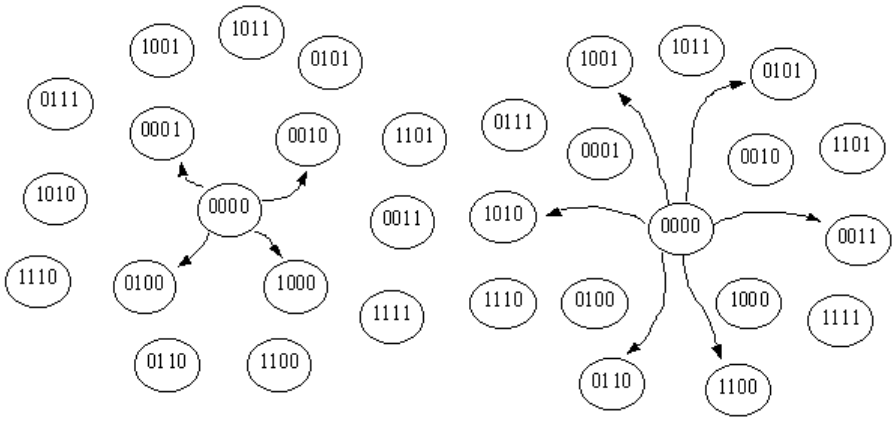


Figura 2.6: *Efecto sobre el espacio de búsqueda ante un cambio de operador*

$$\mathcal{N}_{\mathcal{A}}(x) = \{1000, 0100, 0010, 0001\}$$
$$\mathcal{N}_{\mathcal{B}}(x) = \{1100, 1010, 1001, 0110, 0101, 0011\}$$

Bajo la visión de landscape como un grafo, esta situación se representa en la Fig. 2.6, donde cada nodo de ambos grafos representa un string binario de cuatro elementos. Ahora siendo el nodo $x = 0000$ y $\mathcal{A}$, $\mathcal{B}$, operaciones de modificación como las definidas anteriormente, el grafo de la izquierda representa las soluciones accesibles desde x mediante la operación $\mathcal{A}$. La accesibilidad queda reflejada por la existencia de aristas. Si cambiamos la operación por $\mathcal{B}$, obtenemos el grafo de la derecha, donde se han eliminado las aristas previas y se han dibujado las nuevas que conectan x con aquellos nodos en $\mathcal{N}_{\mathcal{B}}(x)$.

Por lo tanto, se observa que un cambio de operación modifica las aristas del grafo, o espacio de búsqueda, modificando el conjunto de soluciones accesibles. La consecuencia de este enfoque es que un "pico" en un landscape no es necesariamente un "pico" en otro. En otros términos, que una solución sea óptimo local bajo un operador, no implica que lo siga siendo cuando el operador se cambia, y es ésta la razón que justifica la utilización de varios operadores en el proceso de búsqueda.

2.4.2 Justificación Experimental

En la sección anterior se justificó la utilización de diferentes operadores en el proceso de búsqueda bajo el marco de los landscapes. En esta sección se describirán los experimentos realizados para mostrar que dicha estrategia, utilizar varios operadores en el proceso de búsqueda, permite obtener mejores resultados que utilizando un único operador.

Para los experimentos se utilizó *FANS* como una búsqueda local simple con multiarranque. Las pruebas se realizaron sobre 10 instancias del problema de la mochila: 5 instancias aleatorias, con correlación débil entre costos y beneficios, con una única restriccion y 5 instancias de la versión con múltiples restricciones con óptimos conocidos.

Se utilizó un administrador de vecindario $First$, el cual retorna la primera solución que mejore el costo de la actual, utilizando un número máximo de intentos fijado en $maxTrials = n$, donde n es el tamaño de la instancia. La solución inicial se genera aleatoriamente con un único valor en 1.

Como operador de modificación se utilizó el k-BitFlip, el cual complementa el valor de k bits seleccionados al azar. Se implementaron dos administradores de operación $\mathcal{OS}$. El primero, denominado *Decremento*, hace $k_t = k_{t-1} - 1$ cada vez que el Administrador de Vecindario no pueda obtener una solución mejor que la actual utilizando un operador que modificaba k_{t-1} bits. Es decir, suponiendo $maxK = 3$, entonces el algoritmo comienza a progresar con 3-BitFlip. Cuando se estanca, sigue con 2-BitFlip y luego con 1-BitFlip. El segundo administrador, denominado *Incremento*, aumenta el valor de k: $k_t = k_{t-1} + 1$. En este caso, siempre se comienza con 1-BitFlip, se sigue con 2-BitFlip y así hasta $maxK$-BitFlip. Cuando el administrador de vecindario no pueda obtener una solución mejor con $k = 1$ o $k = maxK$, según se hagan decrementos o incrementos respectivamente, se ejecuta el procedimiento $Escape$, el cual genera una nueva solucion aleatoria. Posteriormente, el algoritmo se reinicia desde esta nueva solución.

El objetivo del experimento es mostrar que utilizar un valor de $maxK > 1$ resulta mejor que utilizar $maxK = 1$. La base de comparación son los resultados obtenidos con $k = 1$. Se elige este valor porque el 1-BitFlip es la

	Base	Decremento $maxK$			Incremento $maxK$		
Prob.	1	2	3	4	2	3	4
MR	9.01	2.19	1.95	1.92	3.98	3.85	3.90
ST	6.86	4.58	3.87	3.58	4.73	4.28	4.11
Total	7.94	3.39	2.91	2.75	4.36	4.06	4.01

Tabla 2.1: *Medias del Error en función de maxK y $\mathcal{OS}$ para instancias con una restricción (ST) y con múltiples restricciones (MR).*

única operación que garantiza que todas las soluciones son accesibles desde cualquier punto del espacio de búsqueda.

Por cada instancia, valor de $maxK = \{1, 2, 3, 4\}$, y administrador de operación $\mathcal{OS} = \{incremento, decremento\}$, se realizaron 30 ejecuciones del algoritmo, finalizando cada una cuando se agotaron las 15000 evaluaciones disponibles de la función de costo o se generaron 45000 soluciones. A continuación se presentan los resultados obtenidos.

Análisis de Resultados

Analizaremos los resultados en términos de la media de los errores agrupados por tipo de instancias y en forma global, para cada valor utilizado de $maxK$ y $\mathcal{OS}$. El error se calcula como:

$$error = 100 * \frac{Valor\,de\,Referencia - Valor\,Obtenido}{Valor\,de\,Referencia} \tag{2.4}$$

donde $Valor\,de\,Referencia$ es el óptimo de la instancia para los problemas de la mochila con múltiples restricciones (MR); para la versión clásica (ST), el valor utilizado es la cota de Dantzig (solución para la versión continua)[5].

En la Tabla 2.1 se muestra la media del error para cada valor de $maxK$ y cada $\mathcal{OS}$, discriminado para las instancias ST y MR. En la Figura 2.7 se muestra la información mediante diagramas de caja.

Se puede observar que tanto en forma global, como para cada tipo de instancia en particular, las versiones del algoritmo que usan $maxK > 1$ obtienen

[5]El cálculo de la Cota de Dantzig se describe, por ejemplo en [86]

mejores resultados que los obtenidos con $maxK = 1$. Esto ocurre para los dos esquemas de adaptación de $maxK$ propuestos. También se verifica que para ambos $\mathcal{OS}$'s, un incremento en el valor de $maxK$ permite reducir el error global (indicado en la columna $Total$). Para el caso de decremento, resulta beneficioso comenzar las mejoras con un operador que cambie 4 bits. Cuando con 4 ya no se obtengan mejoras, se reduce a 3, luego a 2 y finalmente se realiza un "ajuste fino" con $k = 1$. La adaptación de k en sentido contrario tambien resulta beneficiosa, aunque para las instancias MR solo aparecen mejoras respecto a $k = 1$ y no entre los valores obtenidos cuando se utiliza $maxK > 1$.

Es interesante analizar cuanto contribuye cada operador a la obtención del resultado final. En la Figura 2.8 se muestran la cantidad de pasos de mejora (en promedio) que permitió obtener cada operador para cada valor de $maxK$. Los valores obtenidos se escalaron en el rango $[1, 100]$ para mejorar la interpretabilidad. Naturalmente cuando $maxK = 1$, todas las mejoras se obtuvieron con $k = 1$. Es a partir de $maxK = 2$ donde el análisis se vuelve interesante.

Analizando los resultados correspondientes al $\mathcal{OS}$ que decrementa k, se observa que cuando $maxK = 2$, practicamente un 95% de la mejora se obtuvo con 2-BitFlip y el restante porcentaje con $k = 1$. Cuando $maxK = 3$, el 80% del progreso se debe al uso de 3-BitFlip. Del restante 20%, casi toda la mejora se realiza con 2-BitFlip. Finalmente cuando $maxK = 4$, el operador 4-BitFlip es el responsable del 70% de los pasos de mejora. La utilización de 3 y 2-BitFlip completa practicamente la optimización aunque aparecen aproximadamente un 5% de mejoras correspondientes a 1-BitFlip.

Utilizando el esquema de incremento para k, se observa que para un valor de $maxK = 2$, hasta un 20% de mejora se puede conseguir con 2-BitFlip una vez que la búsqueda se estancó con 1-BitFlip. Para $maxK = 3$ y 4, los gráficos muestran que hasta un 75% de la mejora corresponde a 1-BitFlip, pero luego los operadores subsiguientes pueden mejorar los óptimos locales encontrados.

Por último en la Tabla 2.2 se muestran los resultados para $e2b$:la media de la cantidad de evaluaciones realizadas para encontrar el mejor valor. Los valo-

	Base	*e2b* en Decremento			*e2b* en Incremento		
Prob.	1	2	3	4	2	3	4
MR	7012.79	6002.11	6092.35	5585.70	6459.27	6353.82	5793.72
ST	6857.25	7654.31	6605.90	6845.97	7800.72	7404.91	7459.72
Total	6935.02	6828.21	6349.13	6215.84	7129.99	6879.37	6626.72

Tabla 2.2: *Medias de la cantidad de evaluaciones realizadas para obtener la mejor solución e2b, en función de maxK, para instancias con una restricción (ST) y con múltiples restricciones (MR)*

res están discriminados en función de $maxK$ y por tipo de instancia. También se incluyen los valores sobre el conjunto total de problemas.

Cuando se utiliza un esquema de decremento, para las instancias con múltiples restricciones se observa una reducción de los valores a medida que aumenta $maxK$. Para las instancias del problema clásico, esta tendencia no se verifica. En este caso, el valor más alto de evaluaciones se alcanza con $maxK = 2$. Luego aparecen los asociados a $maxK = \{1, 4\}$ mientras que el menor valor corresponde a $maxK = 3$.

Si recordamos que las soluciones iniciales y las de reinicialización se generan con un único valor en 1, es razonable pensar que la utilización de valores pequeños de k implicará que se tengan que realizar más pasos de mejora para conseguir "completar" la mochila. Trivialmente, si la solución óptima tuviera $n-1$ elementos, entonces 1-BitFlip necesitaría como mínimo $n-2$ pasos[6] para obtener dicha solución. Este mínimo se reduce a $(n-1)/k$ para el resto de los operadores. Esto explica el comportamiento sobre las instancias MR pero no el mostrado sobre ST.

Para el esquema de adaptación con incrementos de k, los resultados para MR indican que un aumento de $maxK$ permite, no sólo obtener mejores resultados, sino también de forma más rapida o con menos esfuerzo (menores valores de $e2b$). Para las instancias ST, el valor de $e2b$ para $maxK = 1$ es el menor de todos, seguido por $maxK = 3$ y 4. Naturalmente, esta medida del "esfuerzo" no se puede analizar en forma aislada sino teniendo en mente los resultados obtenidos en términos del error. Por lo tanto, creemos que las

[6]La solución inicial ya contiene un elemento

diferencias en la media del error pueden llegar a compensar un posible aumento de las evaluaciones.

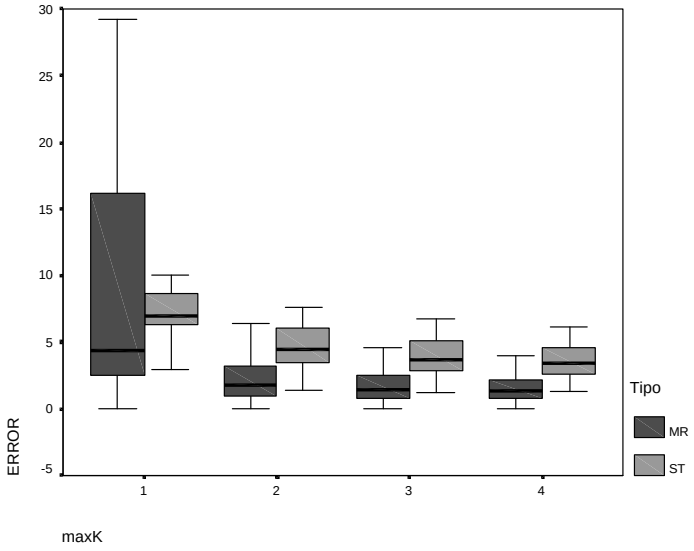
2.5 Conclusiones

En este capítulo se presentó el objeto de estudio de este trabajo: *FANS*. Se describieron las características principales de sus componentes y el esquema del algoritmo.

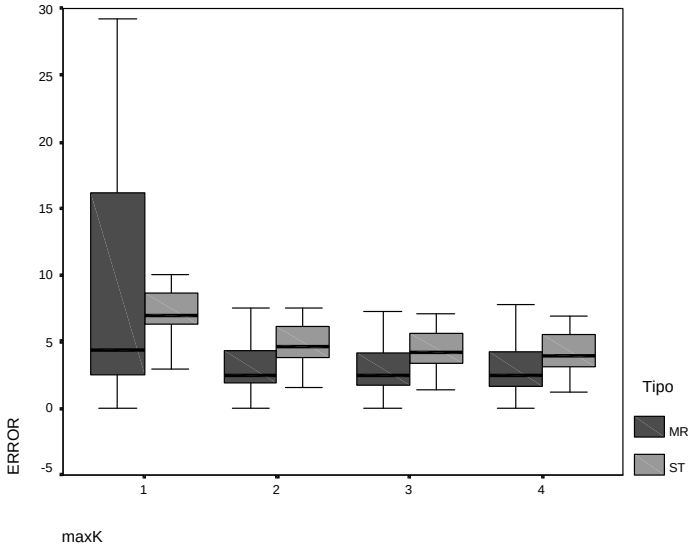
Se mostró como definiciones particulares para la valoración difusa, en conjunción con el administrador de vecindario, permiten obtener o reflejar el comportamiento cualitativo de otros métodos de búsqueda local, lo cual nos llevó a plantear que *FANS* puede considerarse como un framework de heurísticas basadas en búsqueda local. Es decir, mediante una idea relativamente simple, hemos sido capaces de capturar bajo un mismo esquema, un conjunto de métodos que resultan adecuados para resolver aceptablemente una amplia gama de problemas de optimización. También se destacó que *FANS* es en sí misma, una nueva heurística ya que permite obtener comportamientos diferentes a aquellos presentados por otros métodos clásicos.

Se planteó además que *FANS* es un "framework" desde el punto de vista de su implementación y se citaron esquemas similares en este sentido.

Finalmente, y respecto a la utilización de varios operadores, la idea de "landscape" permitió establecer una justificación, al menos "intuitiva" pero clara, de sus potenciales beneficios. Posteriormente se describieron experimentos para mostrar como, aún bajo un esquema muy simple de optimización, el uso de varios operadores permite obtener menores valores de error que los reportados cuando se utiliza un único operador. Por lo tanto, estos resultados dejan abierta la puerta y justifican la investigación de esquemas más sofisticados que utilicen un conjunto de operadores para explorar el espacio de búsqueda, asi como para el diseño de administradores de operación que tengan en cuenta más aspectos del estado de la búsqueda que los utilizados en este capítulo.

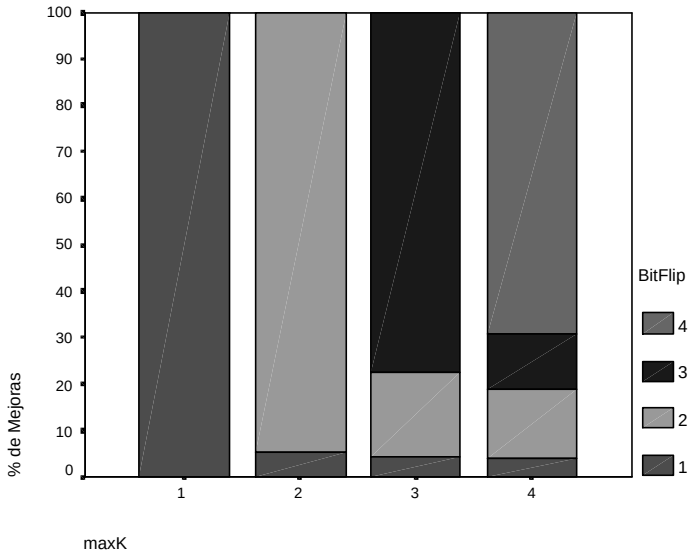


(a)

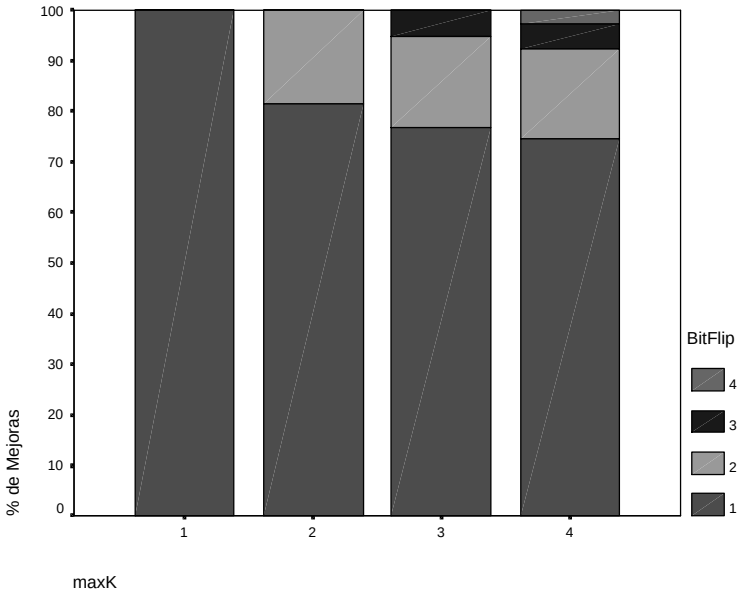


(b)

Figura 2.7: *Intervalos de confianza al* 95% *para las medias del error en función de maxK para instancias con una restricción (ST) y con múltiples restricciones (MR). En (a) resultados con administrador de operación decremento y en (b) con incremento.*



(a)



(b)

Figura 2.8: *Contribución promedio de cada operador BitFlip en las ejecuciones para cada valor de maxK. En (a) resultados con administrador de operación decremento y en (b) con incremento.*

Capítulo 3

Verificación Experimental de *FANS*

En el capítulo anterior se presentaron las características principales de *FANS*, poniendo énfasis en la valoración difusa y en la utilización de varios operadores en el proceso de búsqueda. El paso siguiente que se aborda en este capítulo es verificar la utilidad de *FANS* como herramienta de optimización.

Dado el carácter heurístico de *FANS*, se evaluará su comportamiento en forma empírica a partir de experimentos computacionales sobre un conjunto de instancias de problemas de prueba y se realizarán comparaciones frente a otros algoritmos de propósito general. La comparación empírica de algoritmos es un tema controvertido y que siempre da lugar a debate [13], e incluso se pueden encontrar artículos al respecto con títulos tan sugerentes como los de J. Hooker: *Needed: an Empirical Science of Algorithms* [61] y *Testing Heuristics: We have it all wrong* [62].

Para nuestros experimentos se tendrán en cuenta los siguientes aspectos: cantidad razonable de instancias de prueba en cada problema, implementación propia de todos los algoritmos (salvo 1 caso) y y ejecución de los mismos en la misma plataforma computacional.

Además, se restringirá el análisis a los valores alcanzados de la función de costo, bajo las siguientes condiciones:

- Nulo o escaso conocimiento del dominio del problema, lo que implica la utilización de operadores simples.
- Cantidad limitada e igual de "recursos" para todos los algoritmos. Por recursos se definen evaluaciones de la función de costo y cantidad máxima de soluciones a generar.

Como banco de pruebas se utilizarán los siguientes problemas:

1. el problema de la mochila "clásico" (*ST-KP*, de ahora en adelante),
2. el problema de la mochila con múltiples restricciones (*MR-KP*, de ahora en adelante),
3. el problema de minimización de funciones reales.

Por lo tanto, y con el objetivo de validar la utilidad de *FANS* como herramienta de optimización bajo las condiciones descriptas, se desarrollarán las siguientes tareas: en primer lugar se presentarán las definiciones e instancias de prueba utilizadas en los experimentos para los problemas 1 y 2. Para el problema 1 se utilizarán 45 instancias aleatorias mientras que para el caso 2, las 9 instancias provendrán de un conjunto estandar de prueba. En ambos problemas, *FANS* se comparará frente a un algoritmo genético binario. Posteriormente, se abordará el problema 3, donde *FANS* se comparará frente a un *AG* binario de dominio público y una implementación de recocido simulado, sobre un conjunto de 6 instancias de prueba.

Finalmente, se analizará brevemente *FANS* respecto a la calidad de las soluciones investigadas en el proceso de búsqueda y a la influencia que tienen diferentes administradores de vecindario en los resultados finales.

3.1 *FANS* vs *AG* en Problemas de Mochila

En esta sección se describen los experimentos realizados para evaluar el funcionamiento de *FANS* frente a un algoritmo genético sobre dos versiones del problema de la mochila. En primer lugar se presentan las formulaciones de

los problemas y luego se describen las implementaciones de los algoritmos y los respectivos parámetros. Finalmente, para cada versión del problema se detallan las instancias de prueba utilizadas, las características del experimento realizado y los resultados obtenidos [17, 101].

3.1.1 Formulación de los Problemas de Mochila

A continuación se indica la formulación de las dos versiones del problema de la mochila que se utilizarán en estos experimentos.

Problema de Mochila Estandard

El Problema de la Mochila (*ST-KP*) es un ejemplo clásico en el área de la optimización. Su formulación es la siguiente:

$$max \sum_{j=1}^{n} p_j * x_j \tag{3.1}$$

$$\text{tal que } \sum_{j=1}^{n} w_j * x_j \leq C \tag{3.2}$$

donde n es el número de items, $x_i \in \{0, 1\}$ indica si el item i está colocado en la mochila o no, p_j es el beneficio que reporta la colocación del item i, w_i es el peso del item i y $w_i \in [0..r]$. Finalmente C es la capacidad de la mochila. Como restricciones adicionales se asume que cada item aislado cabe en la mochila ($w_i < C \, \forall i$) y que el conjunto completo de items no cabe en ella ($\sum_{j=1}^{n} w_j > C$). La primera restricción adicional indica que toda solución con un único elemento es factible.

A pesar de su aparente simplicidad, constituye un desafío para muchos algoritmos de búsqueda. Durante los últimos 30 años han surgido algoritmos exactos que permiten abordar la resolución de conjuntos cada vez mas amplios de instancias utilizando tiempos y recursos computacionales razonables. A pesar de ello se plantea que [104]:

> "Mientras que instancias de tipo no correlacionadas con dimensiones hasta n=100000 pueden ser resueltas en menos de un segun-

> do cuando los coeficientes son suficientemente pequeños, ningún algoritmo de la literatura es capaz de resolver instancias fuertemente correlacionadas de tamaño mayor que n=100, cuando los coeficientes crecen."

El lector interesado en profundizar sobre la aplicación de algoritmos exactos y de aproximación sobre este problema, puede consultar el trabajo de Martello y Toth [86], y el de D. Pissinger [104].

Problema de Mochila con Múltiples Restricciones

Dentro de la clase de los problemas de mochila, la versión con Multiples restricciones (*MR-KP*, de ahora en adelante) es la forma más general. Basicamente es un problema de programación lineal entera con la siguiente formulación:

$$Max \sum_{j=1}^{n} p_j * x_j \tag{3.3}$$

$$\text{tal que } \sum_{j=1}^{n} w_{ij} * x_j \leq C_i \text{ con } i = 1..m \tag{3.4}$$

donde m representa el número de restricciones, w_{ij} el costo del item j para la restricción i y los demás elementos son como en el problema clásico.

3.1.2 Implementación de *FANS*

En el momento de aplicar *FANS* sobre un problema específico, se deben proveer definiciones para los componentes. Naturalmente y como ocurre en general, cuanto más dependientes del problema sean las definiciones utilizadas, mejores serán los resultados que se obtengan.

A continuación, se presentan las definiciones particulares para los componentes de *FANS* que se emplearán en los experimentos. Sin duda, muchas otras son factibles, pero las utilizadas fueron elegidas por su simplicidad, teniendo en cuenta la hipótesis de nulo o mínimo conocimiento del problema planteada como condición para la experimentación.

Operador de Modificación k-BitFlip

Este operador selecciona aleatoriamente k posiciones y complementa los valores de bits asociados. No se permite "back mutation".

Valoración Subjetiva: Aceptabilidad

La valoración subjetiva de "Aceptabilidad" que se utilizará representa la siguiente idea: dada una solución actual s, aquellas soluciones vecinas que mejoren su costo tendrán el máximo grado de aceptabilidad. Las soluciones que empeoren un poco el costo, también serán consideradas como aceptables, pero en menor grado. Finalmente aquellas que sean mucho peores que la actual, no serán consideradas como aceptables.

Siendo f la función objetivo, s la solución actual, $q = \mathcal{O}(s)$ una nueva solución, y $\beta < f(s)$ el límite para lo que se considera aceptable, la función de pertenencia siguiente modeliza el comportamiento deseado:

$$\mu(q,s) = \begin{cases} 0.0 & \text{si } f(q) < \beta \\ \frac{(f(q)-\beta)}{(f(s)-\beta)} & \text{si } \beta \leq f(q) \leq f(s) \\ 1.0 & \text{si } f(q) > f(s) \end{cases}$$

La definición de β no es trivial; en este trabajo se propone como primera aproximación la siguiente definición: $\beta = f(s) * (1 - \gamma)$, donde $\gamma \in [0..1]$ representa un factor de escala. Es decir, el máximo nivel de empeoramiento representa un porcentaje del costo actual. Para estos experimentos se utilizó un valor $\gamma = 0.5$.

Como la idea de aceptabilidad se establece en el contexto de cierta solución actual, cada vez que ésta cambie, los parámetros de la valoración serán adaptados. De esta manera, el criterio de aceptabilidad variará en función del estado de la búsqueda.

Administrador de Operación

El administrador de operación producirá variaciones del parámetro k. Comenzando con cierto valor $k = maxK$, cada llamado al administrador decrementará el valor de k en una unidad.

Administrador de Vecindario

Recordemos que se define el vecindario de s como:

$$\hat{\mathcal{N}}(s) = \{s_i \in \mathcal{N}(s) \mid \mu(s_i) > \lambda)\}.$$

donde $\mathcal{N}(s) = \{s_i \mid s_i = \mathcal{O}_i(s)\}$. Es decir, el vecindario a considerar lo forman aquellas soluciones que satisfagan nuestra idea de "aceptabilidad". En otros términos, buscaremos soluciones que pertenezcan al λ-corte del conjunto difuso $\mu()$. Claramente, valores altos de λ implican mayores niveles de exigencia y posiblemente, mayor dificultad para encontrar soluciones con estas características.

Para implementar el administrador de vecindario se proponen las dos estrategias que se describen a continuación.

Esquema First

Este esquema es muy simple: se dispone de un número máximo de intentos ($maxTrials = n$), donde n es el tamaño de la instancia, para encontrar alguna solución que satisfaga el criterio de aceptabilidad.

Esquema $R|S|T$

Como segunda opción, se presenta un Esquema de Agrupamiento Basado en la Calidad [100], o simplemente, Esquema $R|S|T$ cuya idea es: generar R soluciones vecinas "aceptables", agruparlas en S conjuntos, y finalmente elegir y retornar T de ellas. Para las instancias de *ST-KP* se implementa un esquema ($R = n/4|S = 3|T = 1$) y para las de *MR-KP* se utiliza $R = n/2$.

El método funciona de la siguiente manera: dado un operador $\mathcal{O}$, la solución actual s y la función de pertenencia de la valoración difusa utilizada, $\mu()$, el *Generador* intenta construir un conjunto F con R soluciones vecinas "aceptables" en a lo sumo $MaxTrials$ intentos. Este valor se fijó en $n/2$ para *ST-KP* y en n para *MR-KP*, siendo n el tamaño de la instancia. Naturalmente, se verifica que $F \subseteq \mathcal{N}(s)$. Luego, las soluciones $s_i \in F$ obtenidas se agrupan en tres conjuntos difusos, C_1, C_2, C_3 donde cada conjunto contiene soluciones

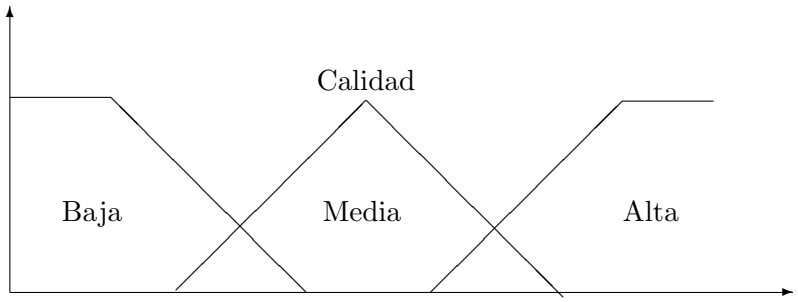


Figura 3.1: *Esquema de Agrupamiento basado en la Calidad*

que verifican $\mu(s_i) \in [min, max]$. Los soportes asociados a cada uno de ellos están dados por los valores de aceptabilidad de las soluciones y son:

$$\begin{aligned} C_1 &= \{s_i \in N(s_i) \mid \mu(s_i) \in (0, 0.4]\} \\ C_2 &= \{s_i \in N(s_i) \mid \mu(s_i) \in [0.3, 0.7]\} \\ C_3 &= \{s_i \in N(s_i) \mid \mu(s_i) \in [0.6, 1.0]\} \end{aligned}$$

Los límites de cada conjunto se transforman linealmente para ajustarse al rango $[\lambda, 1.0]$ siendo λ el mínimo nivel de aceptabilidad requerido.

Estos conjuntos representan los valores {*Baja, Media, Alta*} para la variable lingüistica *Calidad*, con lo cual conseguimos una valoración difusa de las soluciones vecinas aceptables. Podemos considerar a esta etapa como un procedimiento de clustering difuso simple, ya que una solución puede pertenecer a dos clusters y además disponemos del grado de pertenencia a cada uno de ellos. Los conjuntos difusos y sus etiquetas se muestran en la Fig. 3.1.

Una vez construídos los conjuntos o clusters, se debe elegir una solución y retornarla. El criterio implementado se refleja en la siguiente regla:

Retornar alguna solución de las de máxima calidad disponible.

Es decir, se intenta obtener *alguna* solución de *Calidad = Alta* (del conjunto C_3). Si este conjunto estuviera vacío, se intenta obtener alguna solución de *Calidad = Media* (del conjunto C_2) y finalmente de *Calidad = Baja* si fuera necesario. Es posible que C_1 también este vacío. En este caso se retorna la

solución "nula" y se dispara la condición de "vecindario agotado". Es decir, no fue posible encontrar ni un vecino aceptable con los elementos disponibles.

Finalmente, debe notarse que este esquema utiliza, al menos, R evaluaciones de la función de costo por cada llamada al administrador, mientras que en el caso del esquema First el menor valor posible es 1. En ambos casos, el valor $maxTrials$ es la cota superior de número de evaluaciones realizadas en cada llamada.

Procedimiento $Escape()$

Cuando el administrador de vecindario falla en retornar una solución aceptable utilizando el operador con $k = 1$, se considera que la búsqueda está definitivamente estancada. En este caso, se ejecuta un mecanismo de perturbación de la solución actual que consiste simplemente en poner a cero un tercio de las variables en uno. Pensando en una mochila, esto equivale a quitar algunos elementos y dejar todo listo para repetir el procedimiento

3.1.3 Descripción del Algoritmo Genético

El AG utilizado en los experimentos puede considerarse "standard" o tradicional, ya que no incorpora ninguna clase de mecanismos adicionales a los procedimientos de selección, cruce y mutación.

Los individuos se representan mediante codificación binaria con la misma semántica que en *FANS*.

El operador de Mutación es el mismo que el operador de modificación de *FANS*. Dado que ahora no existe el administrador de operación, se utiliza BitFlip con $k = 1$. Respecto al operador de cruce, se implementaron 2 operadores: de 2-puntos y uniforme, dando lugar a 2 algoritmos: *GA2p* y *GAux* respectivamente. Ambas versiones utilizan elitismo: el mejor de cada generación se mantiene en la siguiente.

Para el proceso de Selección se utiliza "Tournament Selection" con $q = 2$ en un esquema $(\mu = popSize/2, \lambda = popSize)$, donde $popSize$ es el tamaño de la población. Es decir, de la población actual se eligen μ individuos por "Torneos" y a partir de estos, se generan λ mediante aplicaciones del operador

de cruce. Si existieran, los individuos no factibles no se agregan al conjunto de entrecruzamiento. Luego, se aplica mutación a todos los individuos (con cierta probabilidad). y si el resultado de la mutación es un individuo no factible (viola restricciones), se hacen hasta 4 intentos más de generar un individuo válido a partir del original. Si en estos 5 intentos no se pudo obtener un individuo factible a través de la mutación, el individuo se deja como estaba.

La población inicial se forma con individuos con una única posición en uno, ya que es el único conocimiento disponible para construir soluciones factibles.

3.2 Problemas de Mochila Estandard

A continuación se detallan las instancias de prueba utilizadas y los experimentos y resultados obtenidos [17].

3.2.1 Instancias de Prueba

Dado que no tenemos conocimiento de la existencia de instancias de prueba standard del problema de la Mochila, recurriremos a las denominadas instancias aleatorias, para generar nuestro propio conjunto de prueba, lo que además nos permitirá analizar los resultados considerando la influencia del tipo de instancia. Para ello, y siguiendo a [86], se obtienen inicialmente los $w_i = U(1, max)$, donde U es un generador uniforme de números aleatorios, y luego en función de la correlación existente entre cada p_i y w_i se generan las siguientes instancias:

- No correlacionadas: $p_i = U(1, max)$,
- Correlación débil: $p_i = U(w_i - t, w_i + t)$, donde t es una constante,
- Correlación fuerte: $p_i = w_i + k$, donde k es una constante.

En general, un incremento en la correlación produce un aumento en la dificultad esperada del problema. Las instancias debilmente correlacionadas son cercanas a problemas reales [86].

Finalmente, la capacidad de la mochila se suele calcular como [86]:

$$c = \alpha * \sum_{i=1}^{n} w_i \text{ con } \alpha \in [0..1] \tag{3.5}$$

Para nuestros experimentos se generaron 5 problemas por cada tipo de instancia y valor de $\alpha = [0.25, 0.5, 0.75]$, dando lugar a un conjunto de 45 instancias de prueba. En general se verifica que cuanto mayor es α, potencialmente más items entrarán en la mochila, ampliando de alguna forma, el espacio de búsqueda. Los restantes valores utilizados fueron los siguientes: $n = 100$, $max = 1000$, $t = max/10$ y $k = 10$.

Dado que el óptimo es desconocido, como valor de comparación se utilizó la cota de Dantzig que representa el valor óptimo para la versión continua del problema.

3.2.2 Experimentos y Resultados

Para los experimentos se generaron 4 versiones del AG y 4 de *FANS* dando lugar a 8 algoritmos. Las versiones del AG surgen de la combinación de dos operadores de cruce (dos puntos (2P) y uniforme (UX)) y dos tamaños de población: 70 y 140 individuos. Para *FANS* las versiones surgen de la combinación de dos administradores de vecindario ($First$ y RST) y dos valores para el parámetro $\lambda = \{0.95, 0.99\}$.

Por cada instancia del problema y algoritmo se hicieron 30 ejecuciones, cada una finalizando al alcanzar las 15000 evaluaciones de la función de costo. Para el análisis de resultados se realizaron comparaciones de los algoritmos por tipo de instancia y en forma global sobre todos los problemas en términos del error, el cual se calcula de la siguiente manera:

$$error = 100 * \frac{Cota\ de\ Dantzig - Valor\ Obtenido}{Cota\ de\ Dantzig} \tag{3.6}$$

De ahora en adelante, utilizaremos la abreviatura NC para designar a las instancias no correlacionadas; DC a aquellas con correlación débil; y FC para las instancias fuertemente correlacionadas.

En primer lugar se analizan los resultados en función del tipo de instancia. La Tabla 3.1 muestra la media y la desviación típica de los errores para cada

	No Corr.		Corr. Debil		Corr. Fuerte	
Algor.	Media	Desv. típ.	Media	Desv. típ.	Media	Desv. típ.
$2P70$	2.35	1.08	2.84	0.92	1.11	1.05
$UX70$	1.96	1.01	2.59	0.93	1.12	1.04
$2P140$	3.44	1.40	3.55	1.01	1.18	1.05
$UX140$	1.80	0.92	2.94	1.03	1.18	1.06
$First0.95$	1.60	0.60	1.85	0.37	0.82	1.01
$RST0.95$	1.76	0.56	2.45	0.63	0.88	1.01
$First0.99$	1.45	0.61	1.03	0.38	0.68	0.99
$RST0.99$	1.51	0.55	1.12	0.37	0.70	0.99

Tabla 3.1: ST-KP*: Media y Desviación Típica de los errores agrupados por tipo de instancia.*

algoritmo en cada tipo de instancia. Cada valor se calcula tomando los errores sobre 15 instancias y sus correspondientes 30 resultados para cada algoritmo (es decir, sobre 450 datos).

El primer elemento a destacar, es que para todos los algoritmos, los valores más bajos de error se alcanzan en las instancias *FC*, lo cual contradice la idea generalizada respecto a que un aumento de correlación produce un aumento en la dificultad. Este es un punto que merece un desarrollo más profundo y que se planteará en trabajos futuros.

Los resultados muestran que para los tres tipos de instancia, las versiones de *FANS* obtienen errores menores en media que los *AG*'s. En general, las diferencias en los valores entre las versiones de *FANS* es pequeña para las instancias *FC* y *NC*. Para el caso *DC*, estas se amplían un poco, obteniendo $RST0.95$ el peor valor. Sin embargo este valor es menor que el mejor obtenido por cualquier *AG*.

Las diferencias en los valores de media entre las versiones de AG son casi inexistentes para las instancias *FC*. Para las *NC* y *DC*, la utilización de cruce uniforme consigue mejorar significativamente los resultados obtenidos con cruce de 2 puntos. En general, las versiones que utilizan una población mayor (140 individuos), resultan peores que las que tienen una población mas pequeña.

Algor.	Media	Desv. típ.	Mediana	Rango
$2P70$	2.10	1.25	1.93	6.52
$UX70$	1.89	1.16	1.65	6.97
$2P140$	2.72	1.60	2.70	7.91
$UX140$	1.97	1.24	1.65	5.62
$First0.95$	1.42	0.83	1.46	4.35
$RST0.95$	1.70	0.99	1.67	4.44
$First0.99$	1.06	0.77	0.88	4.32
$RST0.99$	1.11	0.76	0.97	4.34

Tabla 3.2: ST-KP*: Media, Desviación Típica, Mediana y Rango de los errores sobre los 45 problemas.*

Posiblemente, esto se deba a problemas de convergencia que no son triviales de solucionar. Respecto a las desviaciones típicas, todos los algoritmos obtienen valores reducidos.

La Tabla 3.2 muestra la media, desviación típica, mediana y rango de los errores sobre el conjunto de todos los problemas. En términos de la media, los mejores valores son alcanzados por $RST0.99$ y $First0.99$, seguidos por ambas versiones de *FANS* con $\lambda = 0.95$. Puede observarse que un aumento de λ provoca, para ambos administradores de vecindario, una reducción del error medio. Esto indica que para esta clase de problemas es mejor utilizar una estrategia mas agresiva de búsqueda (implicada por un valor alto de λ). Esto también se verifica en el análisis por tipo de instancia. Cuando se comparan las versiones de *FANS* frente a las del AG, para todos los estadísticos considerados *FANS* obtiene valores menores.

Para confirmar si las diferencias en las medias de los errores eran significativas, se realizaron *t-tests* para cada tipo de instancia y en forma global sobre el conjunto total de problemas. Los resultados aparecen en las Tablas 3.4, 3.5, 3.6 y 3.7, que deben leerse por filas. Un signo '+' en la posición (i, j) indica superioridad del algoritmo i sobre el j a un nivel del 95%. Un signo '-' indica inferioridad y un '=' indica la no existencia de diferencias significati-

	No Corr.			Corr. Debil			Corr. Fuerte		
Algor	0.25	0.5	0.75	0.25	0.5	0.75	0.25	0.5	0.75
$2P70$	3.13	2.45	1.46	3.61	2.91	2.00	1.35	1.55	0.44
$UX70$	2.81	1.92	1.16	3.46	2.52	1.78	1.36	1.56	0.44
$2P140$	4.40	3.66	2.25	4.37	3.71	2.58	1.45	1.61	0.46
$UX140$	2.45	1.90	1.06	3.85	3.01	1.95	1.46	1.61	0.45
$First0.95$	2.04	1.53	1.22	1.81	1.99	1.74	0.88	1.31	0.29
$RST0.95$	1.98	1.80	1.51	1.89	2.89	2.56	0.93	1.38	0.34
$First0.99$	1.96	1.39	1.01	1.39	0.91	0.80	0.60	1.19	0.25
$RST0.99$	1.92	1.46	1.15	1.47	1.00	0.88	0.61	1.23	0.28

Tabla 3.3: ST-KP*: Resumen de la media de los errores discriminado por tipo de instancia y capacidad*

vas. Por cuestiones de diseño de las tablas, el administrador $First$ se indica solamente como F. Salvo algunas excepciones, se confirma que las versiones de *FANS* resultan superiores en media a las del AG, ya sea considerando el tipo de instancia o en forma global.

Para completar el análisis del error, se generaron histogramas para cada algoritmo los cuales se muestran en la Figura 3.2. Los histogramas brindan un panorama más claro del comportamiento de los algoritmos al mostrar todo el espectro de errores obtenidos. El eje Y de los histogramas representa la cantidad de ejecuciones que finalizaron con determinado valor de error, indicado en el eje X. En primer lugar podemos ver cómo la utilización de cruce uniforme $(ux70, ux140)$ disminuye levemente la dispersión de valores respecto al uso de cruce de dos puntos $(2p70, 2p140)$. En cambio, el aumento de población incrementa la dispersión de los puntos. Como se dijo antes, esto también puede ser un indicio de problemas de convergencia del algoritmo.

Respecto a las versiones de *FANS*, se puede ver como para los casos con $\lambda = 0.99$, practicamente todas las ejecuciones finalizan con un error menor al 2%. El uso de dicho valor de λ, produce resultados más concentrados que los que se obtienen con $\lambda = 0.95$.

Para concluir, es interesante analizar la influencia que tiene en el comportamiento de los algoritmos, el nivel α utilizado en el cálculo de la capacidad. La Tabla 3.3 muestra para cada tipo de instancia y capacidad, la media del error de cada algoritmo. Debe notarse que solo existen 5 problemas por tipo de instancia y capacidad, con lo cual no es posible verificar si las diferencias son significativas o no. En general, salvo para las instancias *FC*, se verifica que un aumento en α produce una reducción en la media de los errores para todos los algoritmos. Esto puede estar relacionado con que, a medida que α aumenta, potencialmente ocurrirá lo mismo con la cantidad de items que pueden entrar en la mochila; es decir, disminuye la posibilidad de encontrar soluciones no factibles. Al existir menos soluciones infactibles, el espacio de búsqueda es más suave, presentando menos "baches" y provocando que los algoritmos puedan desplazarse en forma continua y no a "saltos".

Discriminando dentro de cada clase de instancia por el nivel α correspondiente, se puede observar que la media de los errores es menor para las versiones de *FANS* que para las de los *AG* utilizados.

Tabla 3.4: ST-KP*: Resultados de los T-test sobre todos los problemas en conjunto. (ver descripción en el texto)*

	$2P70$	$UX70$	$2P140$	$UX140$	$F0.95$	$RST0.95$	$F0.99$	$RST0.99$
$2P70$	*	-	+	-	-	-	-	-
$UX70$	+	*	+	=	-	-	-	-
$2P140$	-	-	*	-	-	-	-	-
$UX140$	+	=	+	*	-	-	-	-
$F0.95$	+	+	+	+	*	+	-	-
$RST0.95$	+	+	+	+	-	*	-	-
$F0.99$	+	+	+	+	+	+	*	=
$RST0.99$	+	+	+	+	+	+	=	*

	$2P70$	$UX70$	$2P140$	$UX140$	$F0.95$	$RST0.95$	$F0.99$	$RST0.99$
$2P70$	*	-	+	-	-	-	-	-
$UX70$	+	*	+	-	-	-	-	-
$2P140$	-	-	*	-	-	-	-	-
$UX140$	+	+	+	*	-	=	-	-
$F0.95$	+	+	+	+	*	+	-	=
$RST0.95$	+	+	+	=	-	*	-	-
$F0.99$	+	+	+	+	+	+	*	=
$RST0.99$	+	+	+	+	=	+	=	*

Tabla 3.5: ST-KP*: Resultados de los T-test para instancias no correlacionadas (ver descripción en el texto).*

	$2P70$	$UX70$	$2P140$	$UX140$	$F0.95$	$RST0.95$	$F0.99$	$RST0.99$
$2P70$	*	-	+	+	-	-	-	-
$UX70$	+	*	+	+	-	-	-	-
$2P140$	-	-	*	-	-	-	-	-
$UX140$	-	-	+	*	-	-	-	-
$F0.95$	+	+	+	+	*	+	-	-
$RST0.95$	+	+	+	+	-	*	-	-
$F0.99$	+	+	+	+	+	+	*	=
$RST0.99$	+	+	+	+	+	+	=	*

Tabla 3.6: ST-KP*: Resultados de los T-test para instancias con correlación débil. (ver descripción en el texto).*

	$2P70$	$UX70$	$2P140$	$UX140$	$F0.95$	$RST0.95$	$F0.99$	$RST0.99$
$2P70$	*	=	=	=	-	-	-	-
$UX70$	=	*	=	=	-	-	-	-
$2P140$	=	=	*	=	-	-	-	-
$UX140$	=	=	=	*	-	-	-	-
$F0.95$	+	+	+	+	*	=	-	=
$RST0.95$	+	+	+	+	=	*	-	-
$F0.99$	+	+	+	+	+	+	*	=
$RST0.99$	+	+	+	+	+	+	=	*

Tabla 3.7: ST-KP*: Resultados de los T-test para instancias con correlación fuerte. (ver descripción en el texto).*

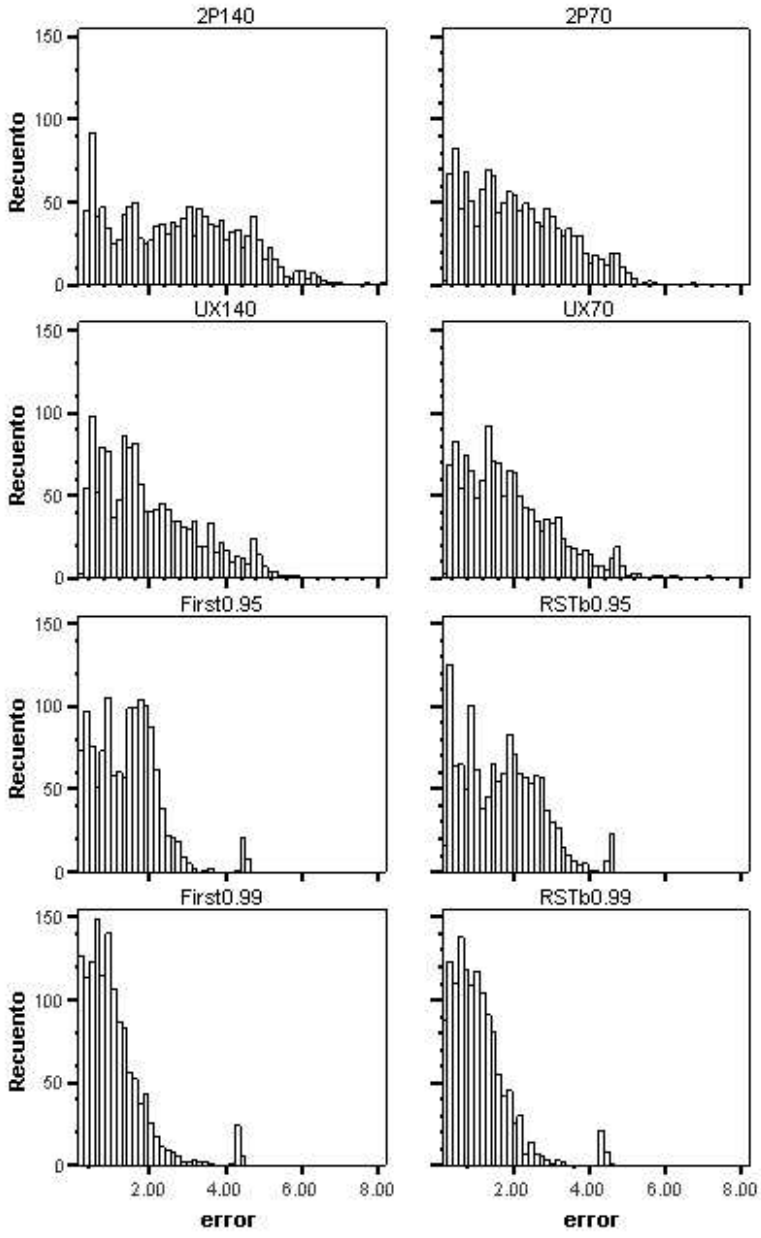


Figura 3.2: ST-KP*: Histogramas del error para cada algoritmo sobre las 45 instancias de prueba.*

Problema	n	m
hp1	28	4
hp2	35	4
pb5	20	10
pb6	40	30
pb7	37	30
sento1	60	30
Weish10	50	5
Weish18	70	5
Weish27	90	5

Tabla 3.8: *Instancias de Problema de Mochila con Múltiples Restricciones*

3.3 Mochila con Múltiples restricciones

En esta sección se detallan las instancias de prueba utilizadas y los experimentos y resultados obtenidos [101].

3.3.1 Instancias de Prueba

Para estos experimentos se utilizarán instancias de prueba pertenecientes a un conjunto de 55 problemas estandard que se encuentran disponibles en [15]. De ese conjunto se utilizaron los 9 problemas que se presentan en la Tabla 3.8, donde n indica la cantidad de variables y m el número de restricciones.

3.3.2 Experimentos y Resultados

Para los experimentos se generaron nuevamente 4 versiones del AG y 4 de *FANS*. Las versiones del AG surgen de la combinación de dos operadores de cruce (dos puntos (2P) y uniforme (UX)) y dos tamaños de población: 100 y 200 individuos. Para *FANS* las versiones surgen de la combinación de dos administradores de vecindario (First y RST) y dos valores para el parámetro $\lambda = \{0.85, 0.95\}$. Luego, por cada instancia del problema y algoritmo se hicieron 30 ejecuciones, cada una finalizando al alcanzar las 15000 evaluaciones

Medias del Error

PROB	$rst85$	$first85$	$rst95$	$first95$	$2p100$	$ux100$	$2p200$	$ux200$
hp1	**1.05**	1.24	1.22	1.16	3.13	2.04	1.82	1.21
hp2	1.46	**1.43**	1.64	2.11	4.91	6.58	4.33	5.68
pb5	0.90	0.99	**0.84**	0.99	1.90	2.32	1.81	1.79
pb6	4.21	4.57	4.66	**4.11**	6.86	6.79	5.52	5.75
pb7	1.89	1.94	1.93	**1.85**	3.48	2.70	2.10	2.19
sento1	1.55	1.41	1.33	1.29	2.04	1.58	1.65	**0.83**
weish10	0.91	0.79	0.79	0.75	0.98	0.80	0.84	**0.60**
weish18	3.34	1.56	0.94	0.94	1.39	0.96	1.52	**0.73**
weish27	4.09	2.86	2.00	2.08	3.08	1.75	3.53	**0.96**

Tabla 3.9: *Medias de los errores, sobre 30 ejecuciones, para cada heurística en cada problema.*

de la función de costo. Dado que las instancias de prueba tienen óptimo conocido, el error se calcula ahora de la siguiente manera:

$$error = 100 * \frac{Optimo - Valor\ Obtenido}{Optimo} \tag{3.7}$$

En la Tabla 3.9 se presenta la media del error para cada algoritmo en cada problema de prueba. La media del error se calcula sobre las 30 ejecuciones realizadas. En negrita se indica el mínimo valor alcanzado para cada problema.

En primer lugar podemos observar como para los primeros 5 problemas, los mejores valores en media se alcanzan con alguna versión de *FANS*. Para los últimos 4, la mejor opción resulta ser $ux200$. También resulta claro, que existen problemas dificiles de aproximar, como pb6, donde ningún algoritmo consigue errores menores al 4%, y otros relativamente fáciles como weish10, donde todos los algoritmos tienen errores menores al 1%.

Resulta interesante destacar que ninguna versión de *FANS* obtuvo errores superiores al 5%, cosa que si ocurrió para todas las versiones del AG. Si consideramos que un problema P está "razonablemente" resuelto por un algoritmo A si este consigue errores menores o iguales al 2%, podemos establecer un

$rst95$	$first85$	$rst85$	$first85$	$ux200$	$2p200$	$ux100$	$2p100$
8	7	6	6	6	5	4	3

Tabla 3.10: *Nro. de problemas por algoritmo que finalizaron con error medio ¡ 2%*

	Estadística del Error			
Algor.	Media	Mediana	Rango	Desv. típ.
$rst85$	2.16	1.60	12.37	1.80
$first85$	1.87	1.40	8.63	1.57
$rst95$	1.71	1.09	12.37	1.65
$first95$	1.70	1.12	8.89	1.49
$2p100$	3.08	2.29	18.17	3.16
$ux100$	2.83	1.77	12.50	3.00
$2p200$	2.57	2.01	13.40	2.46
$ux200$	2.19	1.15	12.50	2.77

Tabla 3.11: *Estadística del error sobre el conjunto de problemas de prueba*

ranking inicial para los algoritmos. Este ranking se muestra en la Tabla 3.10, donde cada columna indica el número de problemas resueltos con error menor o igual al 2% (sobre un total de 9) para cada algoritmo. Bajo estas condiciones, el mejor algoritmo resulta $rst95$, seguido de $first85$. Luego aparecen igualados $rst85$, $first95$ y $ux200$ y finalmente siguen los otros AG's.

Para completar el análisis, se presentan en la Tabla 3.11 los valores de media, mediana, rango y desviación típica del error para cada algoritmo sobre todos los problemas en conjunto. El análisis global muestra que, en media, la peor versión de *FANS* ($rst85$) resulta tan buena como la mejor versión de AG ($ux200$). Los mejores resultados los alcanzan $rst95$ y $first95$. Si consideramos la mediana como un mejor estimador de la calidad, entonces $ux200$ resulta practicamente igual que estos dos algoritmos. Es interesante notar que las diferencias entre media y mediana son mayores para los AG, lo cual indica claramente la existencia de valores extremos en algunos problemas. La desviación típica resulta menor en las versiones de *FANS* que en las de AG.

	$2p100$	$ux100$	$2p200$	$ux200$	$F0.85$	$RST0.85$	$F0.95$	$RST0.95$
$2p100$	*	=	-	-	-	-	-	-
$ux100$	=	*	=	-	-	-	-	-
$2p200$	+	=	*	=	-	-	-	-
$ux200$	=	=	=	*	=	=	-	-
0.85	+	+	+	=	*	=	=	=
$RST0.85$	+	+	+	=	=	*	-	-
0.95	+	+	+	+	=	+	*	=
$RST0.95$	+	+	+	+	+	=	=	*

Tabla 3.12: *T-test en* MR-KP. *Un + en la posición* (i, j) *indica superioridad del algoritmo i sobre el j* ($p < 0.05$). *Un - indica inferioridad y un = indica la no existencia de diferencias significativas. F indica el administrador de vecindario First.*

Para verificar si las diferencias entre algoritmos en terminos de las medias del error sobre el conjunto de pruebas son significativas, se realizaron t-test. Los resultados aparecen en la Tabla 3.12 y permiten verificar que las versiones de *FANS* resultan mejores o iguales que las del AG. *FANS* con $\lambda = 0.85$ permite obtener los mismos resultados que la versión mas costosa del AG: $ux200$. La utilización de $\lambda = 0.95$ con los dos administradores de vecindario evaluados permite obtener dos algoritmos que resultan igualmente buenos entre sí, y superiores a todos los demás esquemas. Para el caso de los AG, y a diferencia de lo que ocurría para los problemas de mochila con una sola restricción, el incremento en el tamaño de la población implicó una mejora significativa de los resultados obtenidos.

Para completar el análisis del error, se generaron histogramas para cada algoritmo los cuales se muestran en la Figura 3.3. El eje Y de los histogramas representa la cantidad de ejecuciones que finalizaron con determinado valor de error, indicado en el eje X. Las diferencias entre las versiones de *FANS* y las del AG resultan claramente visibles. Para el caso de *FANS*, casi todas las ejecuciones de las cuatro versiones obtuvieron resultados por debajo del 5%. Para los AG, los valores se encuentran mucho más dispersos, apareciendo incluso casos con error mayor al 10%.

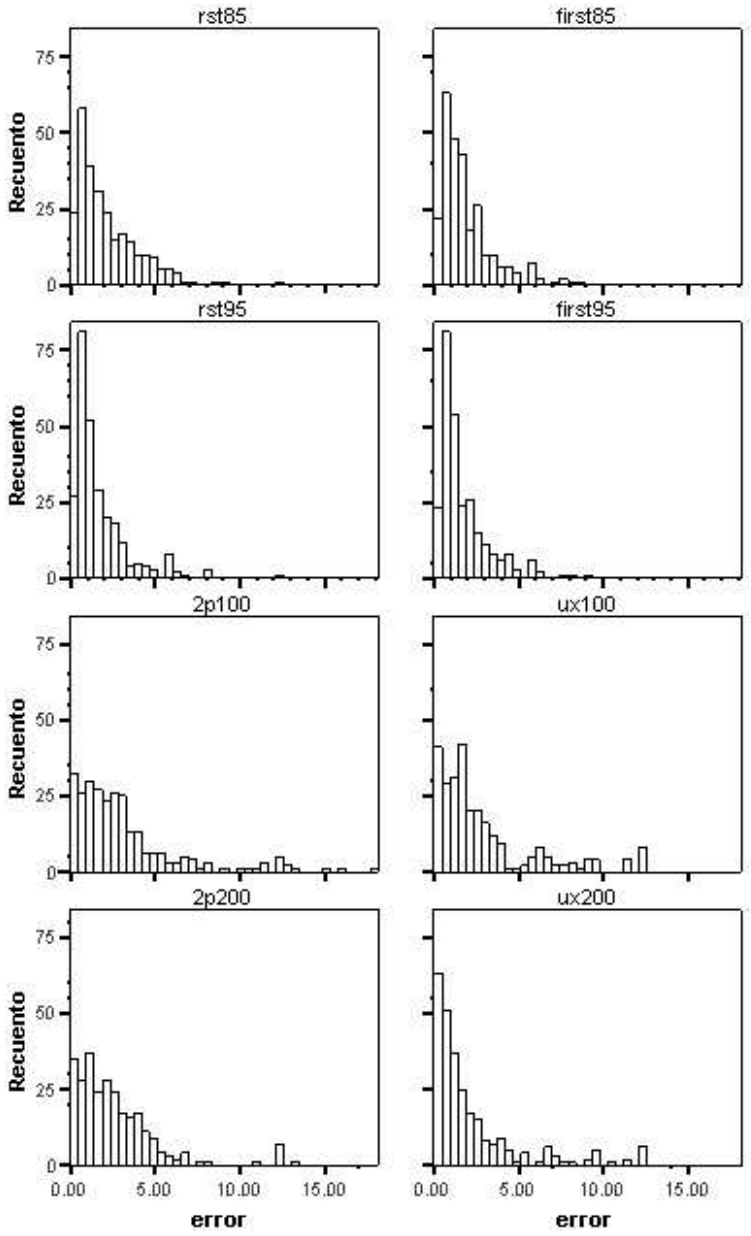


Figura 3.3: MR-KP*: Histogramas del Error para cada algoritmo sobre las 9 instancias de prueba.*

3.4 Experimentos sobre Funciones Reales

En las secciones anteriores se presentó la aplicación de *FANS* sobre dos versiones del problema de la mochila y se compararon los resultados obtenidos frente a otras heurísticas de propósito general. Ahora, en esta sección se muestra como *FANS* también puede ajustarse para resolver problemas de minimización de funciones reales del tipo $f : R^n \rightarrow R$ con n variables.

En primer lugar se describen las funciones de prueba utilizadas junto con sus características. Luego, y como en el caso de los problemas anteriores, se proveen definiciones particulares para cada componente. Finalmente, se presentan los resultados [100, 99, 18].

3.4.1 Funciones de Prueba

Como instancias de prueba se utilizó un conjunto de funciones reales que cubren un amplio espectro de posibilidades en referencia a multimodalidad, separabilidad, etc [9, 90]. En la Tabla 3.13 se muestra la definición de estas funciones y el rango para las variables $x_i \in [min, max]$.

A continuación se presentan algunas características, el valor óptimo $x^* = (x_1, .., x_n)$ y la solución óptima $f(x^*) = k$. A menos que se establezca lo contrario, todas las funciones alcanzan el óptimo en el punto $x^* = (0, .., 0)$ con $f(x^*) = 0$ y $n = 25$.

El modelo de la Esfera, f_{sph}, es una función unimodal, continua y fuertemente convexa. Rastringin, f_{ras}, es una función multimodal, escalable y continua. Rosenbrock, f_{ros}, es continua, unimodal y no separable; constituye un desafío para muchos algoritmos de optimización. Alcanza el óptimo en $x^* = (1, \ldots, 1)$ con $f(x^*) = 0$. Griewank, f_{gri}, es una función continua, multimodal y no separable. Dado que f_{10} posee interacciones no lineales entre las dos variables, su versión extendida ef_{10} induce interacciones no lineales entre múltiples variables. Además es no separable y se utiliza con $n = 10$. Finalmente Schaffer, f_{sch}, es continua, unimodal y presenta dificultades similares a Rosenbrock.

$f_{sph}(\hat{x}) = \sum_{i=1}^{n} x_i^2$	[-5.12, 5.12]
$f_{ras}(\hat{x}) = 10.n + \sum_{i=1}^{n}[x_i^2 - 10 * cos(2\pi x_i)]$	[-5.12, 5.12]
$f_{ros}(\hat{x}) = \sum_{i=1}^{n-1}(100 * (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	[-5.12, 5.12]
$f_{gri}(\hat{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$	[-600,600]
$f_{sch}(\hat{x}) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$	[-65.5, 65.5]
$ef_{10}(\hat{x}) = f_{10}(x_1, x_2) + \ldots + f_{10}(x_{i-1}, x_i) + \ldots + f_{10}(x_n, x_1)$ donde $f_{10} = (x^2 + y^2)^{0.25} * [sin^2(50 * (x^2 + y^2)^{0.1}) + 1]$	[-100.0, 100.0]

Tabla 3.13: *Funciones de Prueba utilizadas*

3.4.2 Implementación de *FANS*

Una vez presentadas las funciones de prueba y siguiendo el esquema de las secciones anteriores, se describen a continuación las definiciones utilizadas para los componentes de *FANS*.

Las soluciones se representan mediante vectores de números reales, donde cada posición representa una variable. Por lo tanto, es necesario definir un operador de modificación específico para esta representación y diseñar del correspondiente administrador. Los demás componentes de *FANS*, como el administrador de vecindario, se mantienen con la definición utilizada para los anteriores problemas de prueba.

El Operador de Modificación $r_k\, Move$

El operador de modificación producirá variaciones en algunas variables. Siendo s la solución actual con n variables $x_1 \dots x_n$, el operador $r_k\, Move$ selecciona al azar $n/2$ variables y les aplica una perturbación aleatoria positiva o negativa. Dado un valor $m \in (0..Top]$, con $Top \in (0..100]$, y una variable $x_i \in [min, max]$, se calcula un "rango de modificación" r de la siguiente manera: $r = (max - min) * m/100$. Es decir, m representa un porcentaje del rango disponible para la variable.

Luego, se calcula el nuevo valor $\hat{x}_i$ para cada variable seleccionada x_i como $\hat{x}_i = x_i + v/10^q$ donde v es un número aleatorio generado en el intervalo $[-r/k, r/k]$, y q es un valor entero aleatorio entre [0..32].

El valor k complementa a m en la restricción del grado de modificación de las variables. El valor q permite controlar la granularidad de las perturbaciones. Valores altos implican movimientos pequeños y valores pequeños, movimientos amplios. El administrador de operación es el responsable de adaptar los valores asignados a m y Top. Naturalmente, cada vez que dichos valores son modificados, el comportamiento del operador variará. Los cambios en Top y m son una de las claves para equilibrar las etapas de exploración/explotación: valores altos de m permiten modificaciones mayores en las variables, o lo que es lo mismo, permite buscar soluciones aceptables en espacios más grandes (exploración). Valores pequeños de m concentran la búsqueda en el entorno de la solución actual (explotación).

La Valoración Difusa

Como en los problemas anteriores, las soluciones generadas en el administrador de vecindario serán evaluadas en términos de la valoración "*Aceptable*". Siendo f la función objetivo, s la solución actual, $q = r_k Move(s)$ una solución vecina, y $\beta > f(s)$ el límite para lo que se considera aceptable, se define la siguiente función de pertenencia:

$$\mu(q,s) = \begin{cases} 1.0 & \text{si } f(q) < f(s) \\ \frac{(f(q)-\beta)}{(f(s)-\beta)} & \text{si } f(s) \leq f(q) \leq \beta \\ 0.0 & \text{si } f(q) > \beta \end{cases}$$

Como primera aproximación, y dado que se trata de problemas de minimización, se optó por $\beta = f(s) * (1 + \gamma)$, donde $\gamma \in [0..1]$. El valor de β será recalculado cada vez que la solución actual cambie.

El Administrador de Operación

Para este ejemplo, dos situaciones diferentes activarán el administrador: cuando no se encuentren soluciones aceptables en el vecindario actual (condición de "vecindario agotado") o cuando hayan transcurrido más de cierto número de iteraciones sin mejoras en la mejor solución encontrada (condición de "estancamiento"). Para ambos casos, la respuesta es la misma: el porcentaje de variación m del operador r_k $Move$ es modificado con la siguiente estrategia: dado un valor inicial $m = m_0$, se provocan decrementos pequeños (0.1) cada vez que se producen las condiciones de activación. Cuando se alcanza $m = 0.1$, cada vez que se produzcan las condiciones de activación, el valor se incrementará (tambien en pasos de 0.1) hasta que $m = Tope$. Luego, el ciclo se repite hasta el final de la ejecución.

Para permitir una etapa de exploración amplia, el 4% inicial de las iteraciones disponibles, se realizan fijando $m = 100$. Agotadas estas iteraciones, el valor de m es adaptado como se describió anteriormente. Cuando se alcanza $m = 3.5$ se hace $Tope = 3.5$ para dejar acotado el valor que podrá alcanzar m en la etapa de incremento.

El valor de $Tope$ también se adapta cuando se da la condición de "estancamiento" y en este caso, la estrategia utilizada producirá un decremento del valor con paso 0.01. Esto implica una reducción en la amplitud de variación de m y por lo tanto, una concentración de la búsqueda. Este esquema de adaptación se muestra de forma gráfica en la Fig. 3.4 donde aparecen las variaciónes para m y $Tope$ en función del número de llamadas al Administrador de Operación.

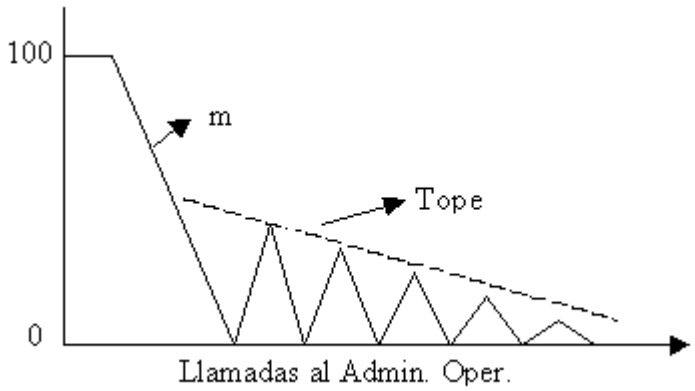


Figura 3.4: *Esquema de Adaptación para los parámetros m y $Tope$.*

El Administrador de Vecindario

Como administrador de Vecindario, y de la misma forma que en los problemas previos, se utilizará el esquema $R|S|T$ con parámetros $R = 3|S = 3|T = 1$ y $maxTrials = 6$.

3.4.3 Experimentos y Resultados

Para evaluar el comportamiento de *FANS* sobre esta clase de problema se realizaron experimentos y comparaciones de *FANS* frente un *AG* binario de dominio público (*AGB*, de ahora en adelante) y una implementación propia de *SA*. A continuación se presentan los parámetros utilizados en cada método, los cuales se fijaron empiricamente a partir de una serie reducida de experimentos preliminares.

Para cada problema, se realizaron 25 ejecuciones de cada algoritmo. Cada ejecución parte de una solución inicial aleatoria diferente y finaliza cuando se alcanza el máximo número de evaluaciones de la función de costo, fijado en $3 * 10^5$. El nivel de exigencia para considerar a una solución como "aceptable" se fijó en $\lambda = 0.95$.

El *AGB* utilizado es el que se presenta en [8] y se establecieron los siguientes parámetros: 60 individuos, 32 bits por variable, cruce uniforme con

	Esfera				Rosenbrock			
	AE	*AB*	*SD*	*BF*	*AE*	*AB*	*SD*	*BF*
BCGA	3.0E+05	2.6E-02	2.1E-02	6.7E-03	3.0E+05	1.1E+02	3.8E+01	2.6E+01
CHC	*	2.0E-32	9.0E-32	5.0E-32	*	2.0E+01	7.0E-01	2.0E+01
SA	3.0E+05	2.0E-08	5.4E-09	9.1E-09	3.0E+05	3.0E+00	4.6E+00	4.0E-03
FANS	3.0E+05	2.6E-26	2.7E-26	3.3E-27	2.9E+05	8.7E+00	1.9E+01	1.8E-04

	Rastringin				Schaffer			
	AE	*AB*	*SD*	*BF*	*AE*	*AB*	*SD*	*BF*
BCGA	3.0E+05	1.6E+01	4.4E+00	7.2E+00	3.0E+05	1.0E+03	3.2E+02	4.1E+02
CHC	*	0.0E+00	0.0E+00	100%	*	1.0E-01	3.0E-01	7.0E-12
SA	3.0E+05	2.4E-08	5.3E-09	1.6E-08	3.0E+05	8.7E-05	9.1E-05	4.3E-06
FANS	1.3E+05	9.6E-01	9.8E-01	4.0E-01	3.0E+05	3.5E-05	4.8E-05	7.9E-07

	Griewank				f10 Extendida			
	AE	*AB*	*SD*	*BF*	*AE*	*AB*	*SD*	*BF*
BCGA	3.0E+05	1.1E+00	1.1E-01	8.9E-01	3.0E+05	1.2E+00	3.6E-01	7.1E-01
CHC	*	0.0E+00	0.0E+00	100%	*	1.0E-07	2.0E-08	9.0E-08
SA	3.0E+05	2.1E-02	2.1E-02	1.9E-08	3.0E+05	1.0E-07	2.5E-08	5.9E-08
FANS	2.0E+05	1.5E-02	1.6E-02	5.4E-20	3.0E+05	2.4E-11	1.7E-11	9.1E-12

Tabla 3.14: *Resultados obtenidos por* FANS*, BCGA y SA.*

probabilidad de cruce $p_c = 0.8$, operador de mutación estandar con probabilidad $p_m = 0.01$ y un mecanismo de selección proporcional. La población inicial se genera de forma aleatoria.

El *SA* utiliza el mismo operador de modificación que *FANS* pero sin adaptaciones (no se utiliza el administrador de operación); El valor del parámetro m del operador se fijó en 100. El esquema de enfriamiento utilizado fue $T(t) = T_0 * \alpha^t$, donde $T(t)$ es la temperatura en la iteración t, La temperatura inicial se fijó en $T_0 = 3.0$ y el factor de decrecimiento se fijó en $\alpha = 0.99993$. La temperatura se adapta después de haber generado 10 soluciones.

Los resultados obtenidos se presentan en la Tabla 3.14, donde AE es el número promedio de evaluaciones de la función de costo realizadas para alcanzar el mejor valor; AB es el promedio de los mejores valores hallados en cada una de las 25 ejecuciones; SD es la desviación estandar de los mejores valores encontrados; BF es el mejor de los valores encontrados en el conjunto de ejecuciones. Si aparece un valor porcentual, este indica el porcentaje sobre las 25 ejecuciones donde el óptimo fue alcanzado.

La Tabla 3.14 muestra claramente que *FANS* es mejor que *AGB* en todas

las funciones, tanto en términos de los valores AB como BF. Este hecho fue verificado mediante test de hipótesis con un nivel de confianza del 95%.

También se verifica que *FANS* es mejor que *SA* en las funciones ef_{10}, f_{sph} y f_{sch}. No se encontraron diferencias significativas (en términos estadísticos) en los valores AB para f_{ros} y f_{gri}; sin embargo debe notarse que los valores BF son mejores en *FANS* para ambos casos. En el caso de f_{ros}, se puede establecer que *FANS* progresa de forma consistente pero más lentamente que *SA*. En f_{gri}, es claro que *FANS* no puede escapar del gran número de mínimos locales existentes. Solamente en 7 de las 25 ejecuciones, *FANS* obtuvo valores del orden de 10^{-20}. Finalmente queda analizar f_{ras}. Mientras los tests de hipótesis muestran que *SA* es mejor que *FANS* en términos de los valores AB, *SA* nunca alcanza el óptimo mientras que *FANS* lo hace en el 40% de las ejecuciones. Incluso este valor pudo ser mejorado hasta el 90% cuando se fijó $maxTrials = 5$ en el administrador de vecindario. Este comportamiento resulta completamente razonable ya que un valor mayor de $maxTrials$ permite visitar más soluciones si es necesario.

En síntesis, en todos los casos *FANS* fue superior al *AGB*, tanto en media como en términos de los valores BF. Respecto a *SA*, *FANS* fue siempre superior en términos de los valores BF, y superior en media en 3 de 6 casos. Solamente consiguió peores resultados en media, en una sola de las funciones de prueba utilizadas.

La Fig. 3.5 muestra la evolución del costo en función del número de evaluaciones para cada algoritmo en cada función de prueba. En todos los casos, el eje Y representa el costo, y el eje X el número de evaluaciones (x 300). Los gráficos permiten reflejar en cierta forma la dinámica de los algoritmos utilizados. Claramente, el *AGB* se ve afectado por convergencia prematura, un problema complejo de solucionar y que no es motivo de análisis en este trabajo. En principio se podría aumentar el tamaño de la población, pero eso implicaría un mayor gasto de evaluaciones de la función de costo por generación lo cual podría provocar una situación de no convergencia.

Analizando las curvas correspondientes a *SA* y *FANS*, se observa que para f_{sph}, f_{gri} y ef_{10}, *FANS* converge a mejores valores y en forma más rapida. En

f_{ras} y f_{ros}, el SA presenta un comportamiento mejor. En el primer caso, puede observarse como *FANS* queda rapidamente atrapado en un mínimo local; en el segundo, se observa como, luego de un comienzo equilibrado, *FANS* se estanca y en SA se produce una mejora adicional. El gráfico correspondiente a f_{sch}, muestra como *FANS* progresa en forma más lenta que SA durante casi toda la simulación, superandolo al final de la misma.

Para finalizar el análisis, en la Tabla 3.14 tambien se incluyen resultados obtenidos mediante el algoritmo CHC [36] y presentados en [56]. En dicho trabajo, los autores presentaban su algoritmo y utilizaban el CHC como medida de comparación. El CHC es un algoritmo genético con codificación real especialmente diseñado para resolver los problemas derivados de la convergencia prematura y que es usualmente utilizado como punto de referencia. En el trabajo citado, CHC fue ejecutado 30 veces, finalizando cada una al alcanzar las $5 * 10^5$ evaluaciones de la función de costo . La Tabla muestra que esta versión relativamente simple de *FANS* supera claramente al CHC en las funciones f_{sch}, f_{ros} y ef_{10} en términos de los valores AB. Respecto a la función f_{sph}, creemos que el nivel de precisión alcanzado por *FANS* (10^{-26}) es suficiente para cualquier aplicación práctica. Los resultados del CHC sobre f_{ras} y f_{gri} son excelentes ya que siempre alcanza el óptimo correspondiente.

Aunque los resultados de *FANS* sobre la función f_{gri} no son buenos, los valores obtenidos resultan comparables frente a los obtenidos mediante AG's distribuidos también presentados en [56].

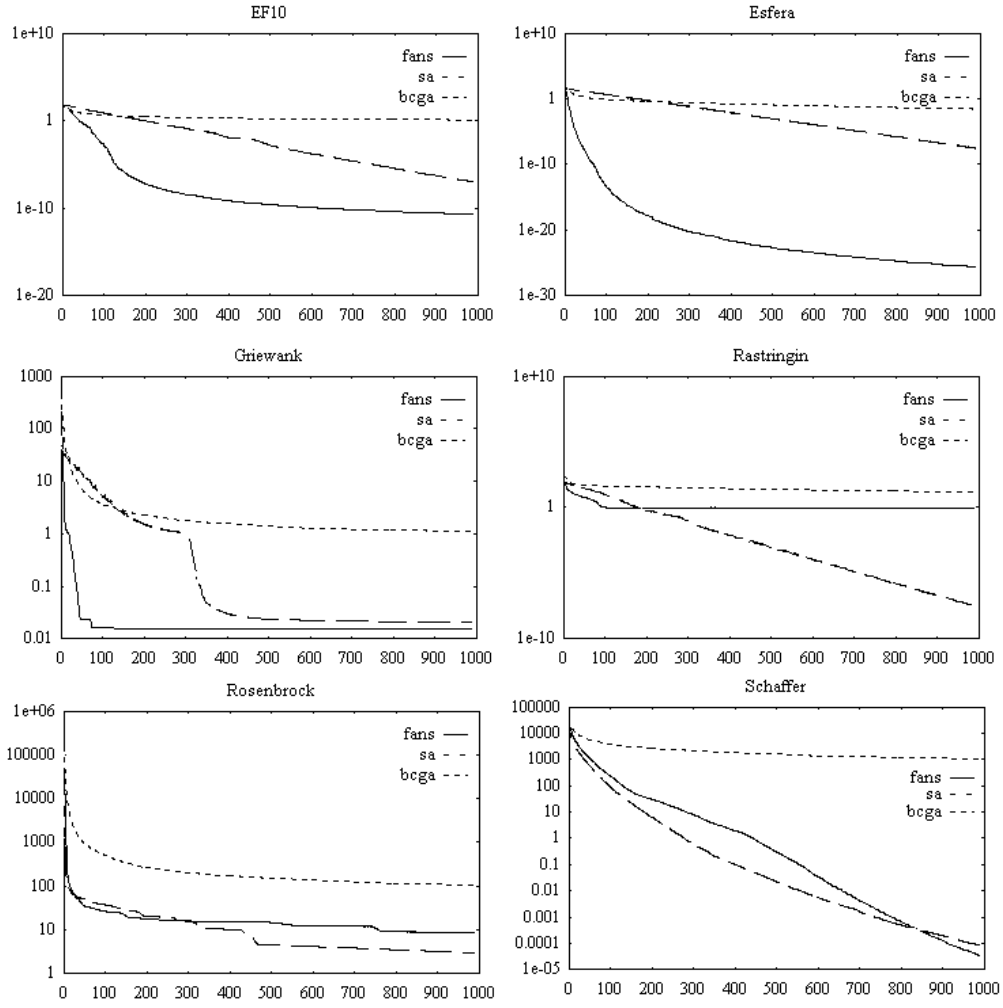


Figura 3.5: *Evolución del costo respecto a las evaluaciones (x 300) para AGB,* FANS *y* SA *en cada función de prueba.*

3.5 Análisis Complementarios de *FANS*

En las secciones anteriores, se han presentado resultados que comprueban que *FANS* es una herramienta útil para resolver problemas de optimización de diversos tipos.

El objetivo de esta sección es analizar dos aspectos del funcionamiento interno de *FANS*. En el primero, se evaluará la calidad de las soluciones generadas y utilizadas por *FANS* durante el proceso de búsqueda en el problema de la mochila con una y varias restricciones. El segundo aspecto a considerar el impacto que tienen diferentes administradores de vecindario en el comportamiento de *FANS*. Ambos aspectos se analizarán a partir de los resultados obtenidos mediante una serie de nuevos experimentos.

3.5.1 Análisis de la Calidad de las Soluciones Investigadas

Un elemento importante en todos los métodos de búsqueda por entornos es el referido a la calidad de las soluciones investigadas, un hecho que está en relación con el operador de modificación utilizado. En el contexto de *FANS*, podemos considerar tres grupos de soluciones: las generadas, las factibles y las aceptables. Parece razonable considerar que la cantidad de soluciones en cada grupo cambiará en función del valor de exigencia λ utilizado.

Con este objetivo se desarrollaron experimentos para indagar en esta relación. Como los valores son independientes de la regla de selección utilizada en el administrador de vecindario, los experimentos se realizaron con la versión de *FANS* que utiliza el esquema RST, con parámetros $5|3|1$ y $maxTrials = 12$. Esto permite un nivel adicional de análisis de las soluciones aceptables en términos de Alta, Media y Baja aceptabilidad. Además, analizaremos la relación entre el valor de λ y la media del error obtenido.

Para los experimentos, se eligieron, 2 instancias de *MR-KP* y 2 instancias de *ST-KP*, y por cada problema, se ejecutaron 25 ejecuciones por nivel de λ con $\lambda = \{0.25, 0.5, 0.75, 0.9, 0.99\}$. Cada ejecución finaliza cuando se realizaron 15000 evaluaciones de la función de costo, o se generaron 22500 soluciones (factibles o no).

Analizaremos primero la variación del error en función de λ. Los resultados

λ	*MR-KP*	*ST-KP*	Total
0.25	4.28	6.61	5.45
0.50	3.72	6.10	4.91
0.75	2.53	5.27	3.9
0.90	2.49	3.64	3.06
0.99	2.85	2.22	2.54

Tabla 3.15: *Media de los Errores en función de λ. Análisis por tipo de Problema.*

obtenidos se presentan en la Tabla 3.15. Es posible observar que para la clase de problemas *ST-KP*, un aumento en λ produce un descenso en la media del error obtenido. Para *MR-KP*, la utilización de $\lambda = 0.99$ incrementa el error respecto a la utilización de $\lambda = \{0.9, 0.75\}$. En principio, estos resultados eran esperables ya que un aumento en λ reduce la posibilidad de elegir soluciones demasiado peores que la actual. Recordemos que en el caso extremo de $\lambda = 1.0$, el administrador de vecindario solo considerará como aceptables aquellas soluciones que mejoren el costo actual.

En la Tabla 3.16, se presentan los resultados obtenidos para cada instancia de cada problema. Las dos primeras columnas son instancias *ST-KP* y las dos últimas corresponden a *MR-KP*. Salvo en los dos últimos problemas, se verifica que un aumento en λ produce una disminución en la media del error obtenido. Existe una mejora muy importante al pasar de 0.75 a 0.99.

Es interesante analizar los casos "negativos". Para el problema Weing3, el comportamiento verificado es exactamente inverso. A mayor valor de λ, peor tasa de error. El mejor valor se alcanza en $\lambda = 0.25$. Sin dudas esto es consecuencia de la estructura intrínseca de la instancia, la cual provoca que sea difícil encontrar soluciones factibles (y además aceptables). Por lo tanto, es razonable que un valor bajo de λ que permite mayor "movilidad" sea adecuado en este caso. Para el problema Weish27, el comportamiento observado es similar al resto, salvo una desmejora para $\lambda = 0.99$.

Analizaremos ahora los datos relacionados con los vecindarios. En la Tabla 3.17 se presentan los resultados obtenidos. Las 3 primeras columnas, indican

λ	wc24	wc27	weing3	weish27
0.25	5.69	7.54	1.14	7.42
0.50	5.09	7.12	1.37	6.07
0.75	4.52	6.03	1.26	3.81
0.90	3.50	3.78	1.73	3.26
0.99	1.61	2.85	2.04	3.67

Tabla 3.16: *Media de los Errores en función de λ. Análisis por Problema.*

la cantidad de soluciones generadas, factibles y aceptables consideradas en el administrador de vecinos. Recordemos que sólo las soluciones factibles son evaluadas y que $Aceptables \subseteq Factibles \subseteq Generadas$.

En primer lugar, se observa un descenso del número de soluciones aceptables respecto al incremento en λ. Esto resulta razonable si pensamos que valores altos de λ inducen vecindarios más restringidos; una vez que tenemos una buena solución deseamos obtener otras *muy* parecidas pero nuestro nivel de exigencia es muy alto y llegado un punto no es posible satisfacerlo.

No resulta simple de comprender por qué se produce una disminución en las soluciones factibles en los problemas *MR-KP* y *ST-KP*, pero una posible explicación es la siguiente: en el análisis del error mostramos como en general se verificaba que un valor mayor de λ producía mejores soluciones (menor tasa de error). Es posible que una vez alcanzadas estas buenas soluciones, no sea fácil obtener soluciones factibles a partir de pequeñas modificaciones como las que provoca el operador utilizado. Podemos suponer que una buena solución es una mochila con un gran número de agregados y que cualquier operación de inserción, viola las restricciones. Por lo tanto, la única operación válida es eliminar algún elemento, pero eso puede llevar a soluciones que no verifiquen el nivel de aceptabilidad. En consecuencia, el administrador no es capaz de encontrar soluciones aceptables (y quizás, ni siquiera factibles).

Recordemos que esta situación provoca que *FANS* ejecute el administrador de operación y de esta forma, el administrador de vecinos intenta encontrar soluciones aceptables mediante la utilización de diferentes operaciones. Estos in-

tentos pueden ser infructuosos muchas veces con el consiguiente "desperdicio" de soluciones generadas. Después de varios intentos fallidos, *FANS* ejecuta el procedimiento para escapar de ese mínimo local. Graficamente, esta situación queda bien reflejada en la Fig. 3.6 donde pueden observarse algunas ejecuciones de *FANS* sobre los problemas Weish27 y WC24 para $\lambda = 0.99$. Los gráficos muestran la evolución del beneficio (cuanto más grande, mejor) en función de las evaluaciones. Claramente pueden detectarse zonas de estancamiento antes de los descensos bruscos provocados por la activación del mecanismo de reinicialización. En dichas zonas de estancamiento, *FANS* intenta encontrar soluciones aceptables mediante diferentes operadores sin tener éxito. Si bien los descensos son "profundos", también se observa que rapidamente se vuelven a alcanzar valores razonables de la función de costo.

Por último evaluaremos la calidad de las soluciones aceptables obtenidas. Estos valores se encuentran en las tres últimas columnas de la Tabla 3.17. En ellas se muestran la cantidad de soluciones de Alta, Media y Baja aceptabilidad encontradas durante las simulaciones. El valor que aparece en soluciones Aceptables se obtiene de la suma de estas tres últimas columnas.

El primer elemento a destacar es que en todos los casos, las soluciones de Alta aceptabilidad son más que las de Media y Baja juntas. También se puede observar como un aumento en λ produce una disminución en todas las categorías. La causa de este descenso fue analizada previamente con las soluciones Aceptables.

Está claro que la cantidad de soluciones en cada categoría decrece de forma diferente; como dato interesante, resulta que con $\lambda = 0.99$ casi desaparecen las soluciones de Media y Baja aceptabilidad. Una posible explicación surge al considerar que las soluciones de dichas categorías siempre empeoran el costo actual. Al utilizar un valor tan alto de λ los pasos de empeoramiento deben ser muy pequeños para poder ser aceptables y en realidad es posible que estos cambios no sean factibles. Por lo tanto creemos que gran parte de las soluciones de Alta aceptabilidad, son en realidad soluciones que mejoran el costo de la solución actual. Sin duda esta situación también está condicionada por el tipo de operación utilizada y la estructura de cada instancia en particular.

Prob	λ	Generadas	Factibles	Aceptables	Alta	Media	Baja
	0.25	22500	10109	7124	4873	1282	970
	0.5	22500	9435	5547	3549	1222	776
MR-KP	0.75	22500	7979	3023	1647	751	624
	0.9	22500	5568	576	417	50	108
	0.99	22500	4993	211	206	2	2
	0.25	22500	11237	10560	5288	2261	3012
	0.5	22500	10349	6840	3805	1526	1508
ST-KP	0.75	22500	9265	3559	2123	660	777
	0.9	22500	8385	1305	847	240	218
	0.99	22500	7902	302	291	5	5

Tabla 3.17: *Media del número de soluciones por grupo en función de* λ.

3.5.2 Evaluación de Administradores de Vecindarios

En esta sección se presentan los experimentos realizados y los resultados obtenidos para comprobar como varía el comportamiento de *FANS* cuando se utilizan diferentes administradores de vecindario. Los experimentos comparan cuatro administradores de vecindario en los cuales el mecanismo de generación de soluciones se mantiene fijo. Los administradores son:

1. F_{rst} : es el utilizado hasta ahora con los siguientes parámetros 5|3|1 y $maxTrials = 12$.

2. $Best_{\mu}$: donde se devuelve aquella solución entre las generadas que tenga el valor más alto de Aceptabilidad $\mu()$. Si hubiera mas de una, se elige al azar.

3. F_{ff} : se retorna la primer solución que sea aceptable

4. $Best_f$: donde se devuelve aquella solución que minimice/maximice f. Si hubiera mas de una, se elige al azar.

Los administradores $Best_f$ y $Best_{\mu}$ intentan obtener 5 soluciones aceptables en un máximo de $maxTrials = 12$ intentos. Entre las obtenidas, aplican su

regla de decisión. F_{ff} dispone hasta $maxTrials = 12$ intentos para conseguir una solución.

Para comprender el funcionamiento, analizaremos con un ejemplo el nivel de explotación de cada administrador. Dada una solución actual s, supongamos que se genera el siguiente conjunto de soluciones aceptables:

$$\mathcal{N}(s) = \{(s_1, 0.8), (s_2, 0.95), (s_3, 0.99), (s_4, 1), (s_5, 1)\}.$$

La segunda componente de cada par indica el valor de μ asociado. Además supongamos que el orden de aparición en el conjunto es también el orden en el que fueron generadas. Veamos el comportamiento de cada administrador:

- Asumiendo que s_3, s_4, s_5 pertenecen al conjunto de "Alta" aceptabilidad, entonces el esquema F_{rst} retornará *alguna* de ellas.
- El esquema F_{ff} retornará s_1 ya que es la primera solución aceptable que se encuentra. Este esquema "no conoce" ninguna de las otras soluciones
- $Best_\mu$ retornará s_4 o s_5 ya que ambas presentan el máximo valor de μ
- $Best_f$ retornará aquella $\hat{s} \in \{s_4, s_5\}$ que maximice/minimice f

Además, merecen consideración los siguientes elementos:

1. Si la solución que maximiza μ es única, entonces la solución retornada por $Best_\mu$ coincide con la retornada por $Best_f$.
2. Si existe más de una solución con $\mu = 1$ (soluciones que mejoran el costo actual), entonces la mejora obtenida por $Best_\mu$ es menor o igual que la obtenida con $Best_f$.
3. Si no existe solución con $\mu = 1$ (todas las soluciones empeoran el costo actual), entonces el empeoramiento obtenido por $Best_\mu$ es mayor o igual que el obtenido con $Best_f$.
4. $Best_f$ permitirá obtener la mayor mejora y el menor empeoramiento.

	F_{rst}		$Best_{\mu}$		F_{ff}		$Best_{f}$	
	Media	DT	Media	DT	Media	DT	Media	DT
ST-KP	1.22	0.63	0.80	1.40	0.66	1.20	1.44	0.82
MR-KP	1.30	1.29	1.33	1.22	1.24	1.21	1.31	1.22

Tabla 3.18: *Media y Desviación Tipica de los Errores obtenidos por* FANS *en función del Administrador de Vecindario utilizado*

5. Si se considera la cantidad de evaluaciones necesarias por iteración, entonces en el mejor caso los administradores $Best_f$, $Best_{\mu}$ y F_{rst} utilizarán 5 evaluaciones (las necesarias para obtener las $R = 5$ soluciones), mientras que F_{ff} utilizará solamente una. Si en todos los casos, el criterio de parada del algoritmo es agotar un número predeterminado de evaluaciones de la función de costo, entonces las ejecuciones de *FANS* con el administrador F_{ff} utilizarán mas iteraciones que las de *FANS* con los otros esquemas.

Teniendo en cuenta estos elementos, podemos establecer el siguiente orden en términos de nivel de explotación:

$$F_{ff} \leq F_{rst} \leq Best_{\mu} \leq Best_{f}$$

y nivel de exploración

$$F_{ff} \geq F_{rst} \geq Best_{\mu} \geq Best_{f}$$

Experimentos y Resultados

Con el objetivo de evaluar el comportamiento de los administradores sugeridos, se realizaron experimentos utilizando 6 instancias de *MR-KP* y 6 instancias de *ST-KP*. Por cada versión de *FANS* con su correspondiente administrador y por problema, se ejecutaron 25 simulaciones con $\lambda = 0.99$. Debe notarse que las soluciones a considerar son independientes del administrador utilizado. Solo cambia la regla de selección de la próxima solución.

En la Tabla 3.18 se muestran los valores correspondientes a la media de los errores y desviación típica para cada tipo de problema; es decir, en cada

caso, son los valores calculados a partir de 6 instancias y 25 ejecuciones de cada versión de *FANS*. En principio, los resultados obtenidos no permiten establecer claramente cual es la mejor opción. Si analizamos los resultados en *ST-KP*, los mejores valores se obtienen con F_{rst} y F_{ff} . Observando la Tabla 3.19, se detecta que ambos esquemas presentan diferencias significativas respecto a $Best_f$ y $Best_\mu$. Debe notarse que el espacio de búsqueda asociado con *ST-KP* presenta algunos "baches" ya que existen soluciones no factibles. Por lo tanto, es necesario utilizar un esquema donde exista un nivel relativamente alto de exploración para poder evitarlos o "saltarlos". Un esquema de explotación como $Best_f$ "completará" la mochila rapidamente pero fallará al no considerar otras configuraciones. Lo mismo puede establecerse de $Best_\mu$.

Para los problemas *MR-KP* no existen diferencias significativas entre los esquemas, aunque el valor mas bajo de error se alcanza en F_{ff} . En este caso el espacio de búsqueda es el más reducido, pero también resulta más difícil obtener soluciones factibles, con lo cual *FANS* se verá forzado a ejecutar varias veces el mecanismo de reinicialización induciendo automaticamente la capacidad de exploración independientemente del administrador utilizado.

ST-KP

	F_{rst}	F_{ff}	$Best_{\mu}$	$Best_f$
F_{rst}		=	+	+
F_{ff}	=		+	+
$Best_{\mu}$	-	-		=
$Best_f$	-	-	=	

MR-KP

	F_{rst}	F_{ff}	$Best_{\mu}$	$Best_f$
F_{rst}		=	=	=
F_{ff}	=		=	=
$Best_{\mu}$	=	=		=
$Best_f$	=	=	=	

Tabla 3.19: *T-test para Prueba de Administradores. Un + en la posición (i, j) indica superioridad del algoritmo i sobre el j a un nivel del 95%. Un - indica inferioridad y un = indica la no existencia de diferencias significativas.*

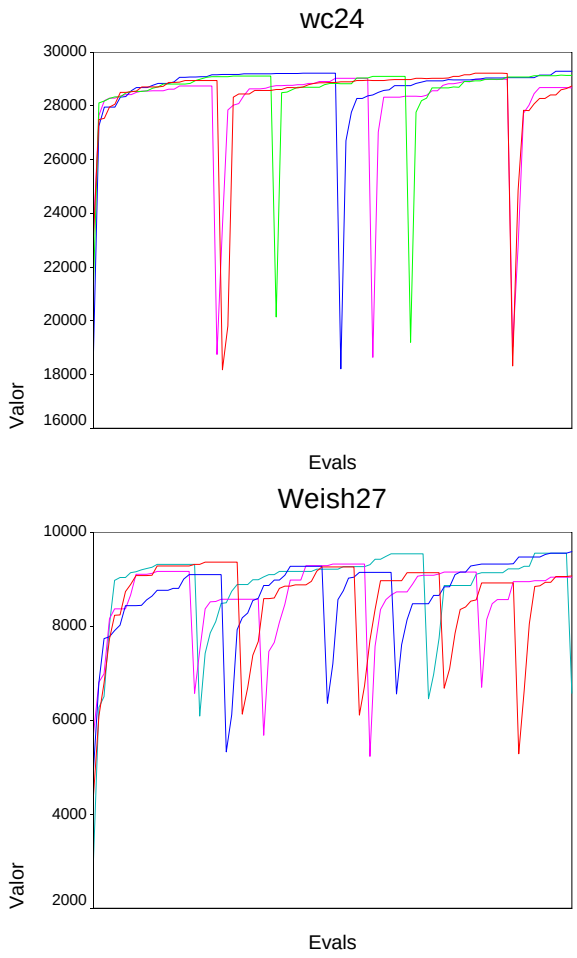


Figura 3.6: *Algunas ejecuciones de* FANS *con* $\lambda = 0.99$. *Se muestra la evolución del beneficio (eje Y) respecto al número de evaluaciones de la función de costo (eje X).*

3.6 Conclusiones

En este capítulo se realizaron experimentos para verificar la utilidad de *FANS* como herramienta de optimización bajo las condiciones de nulo o escaso conocimiento del problema y cantidad limitada de recursos. Además, y para asegurar que las comparaciones fueran justas, se utilizaron implementaciones propias de los algoritmos (salvo el *AGB* en el problema de minimización de funciones reales) y no se realizaron comparaciones respecto a la velocidad de los mismos. Los resultados se analizaron en términos de los errores obtenidos respecto a un valor de referencia.

Las comparaciones de *FANS* frente a un *AG* binario en el problema de la mochila estandar permitieron comprobar que las versiones de *FANS* evaluadas resultaron superiores en media a las del *AG*, tanto en el análisis por tipo de instancia como en el realizado sobre el conjunto global. Respecto a las desviaciones típicas, todos los algoritmos obtienen valores reducidos. Las diferencias en los valores de media entre las versiones de *AG* fueron casi inexistentes para las instancias *FC*. Para las *NC* y *DC*, el uso de cruce uniforme permitió mejorar significativamente los resultados obtenidos con cruce de 2 puntos. En general, las versiones que utilizaron una población mayor (140 individuos), resultaron peores que las que tienen una población mas pequeña.

Respecto a los experimentos sobre el problema de la mochila con múltiples restricciones, las versiones de *FANS* utilizadas resultaron mejores o iguales en media que las del *AG*. *FANS* con $\lambda = 0.85$ permitió obtener los mismos resultados que la versión mas costosa del *AG*: $ux200$. La utilización de $\lambda = 0.95$ con los dos administradores de vecindario evaluados permitió obtener dos algoritmos que resultaron igualmente buenos entre sí, y superiores a todos los demás esquemas. Para el caso de los *AG*, y a diferencia de lo ocurrido para los problemas de mochila con una sola restricción, el incremento en el tamaño de la población implicó una mejora significativa de los resultados obtenidos.

La utilización de *FANS* en la minimización de funciones reales también resultó satisfactoria. Los valores obtenidos por *FANS* resultaron mejores que los obtenidos por un *AGB* en todos los casos y comparados frente a *SA*, solo resultaron peores en media en una sola de las funciones de prueba. Sin em-

bargo, en este caso, *FANS* pudo alcanzar el óptimo de la función en varias ejecuciones mientras que *SA* no lo alcanzó nunca.

Como validación adicional, los resultados de *FANS* se compararon frente a los obtenidos mediante el CHC. En 4 de 6 casos, los resultados fueron mejores o iguales y en dos casos peores. Sin embargo, en estos dos casos los resultados obtenidos por *FANS* resultaron comparables a otros obtenidos mediante versiones distribuidas de algoritmos genéticos.

A partir del teorema de "No Free Lunch" [120], se sabe que no existe un algoritmo que sea mejor que todos los demás sobre todos los problemas de prueba y bajo todas las condiciones posibles. Por ello, podemos afirmar que bajo las condiciones de experimentación propuestas, el conjunto de prueba utilizado y los algoritmos comparados, los resultados obtenidos por *FANS* pueden considerarse como altamente satisfactorios y establecen bases firmes para continuar investigando, profundizando y mejorando el algoritmo .

Finalmente se hizo un análisis inicial respecto a dos aspectos del funcionamiento interno de *FANS*. Se realizaron experimentos para evaluar la calidad de las soluciones generadas y utilizadas en relación con el parámetro λ y a continuación, se propusieron y evaluaron diferentes administradores de vecindario. Para estos administradores se definió un orden en términos de los niveles de exploración/explotación que inducen.

En general, se verificó que un aumento en el valor del parámetro λ produce una disminución en la media del error. Analizando invidualmente cada instancia de prueba, se ha detectado un caso donde el comportamiento es exactamente inverso. Respecto a las comparaciones entre los administradores de vecindario propuestos, solo cabe decir que los experimentos realizados indican (como era esperable), que cada instancia (no problema) requiere un nivel de exploración/explotación diferente. Teniendo en cuenta este aspecto, creemos que disponer de una herramienta versátil como *FANS* cuyo comportamiento se puede modificar en forma simple, resulta ideal para tratar con situaciones de este tipo.

Capítulo 4

Aplicación de *FANS* a Problemas de Bioinformática

Los dos capítulos anteriores estuvieron dedicados a presentar *FANS* y a analizar su comportamiento de forma empírica sobre un conjunto de instancias de problemas de prueba conocidos. Ahora, en este capítulo, evaluaremos los resultados que obtiene *FANS* sobre problemas relevantes del área de la Bioinformática.

En esta memoria consideramos la Bioinformática como un área en la frontera entre la biología y las ciencias de la computación cuyo principal objetivo es el desarrollo y uso de técnicas matemáticas y computacionales para ayudar en la resolución de problemas de la biología molecular. Es especialmente atractiva puesto que las modelizaciones asociadas a gran parte de los problemas de biología que se utilizan, resultan ser NP-Completos y por lo tanto, deben ser abordados mediante técnicas heurísticas. Por lo tanto, la resolución de estos problemas y la obtención de nuevos mecanismos de solución tienen y tendrán impacto, tanto en Informática como en Biología.

Aunque sus raíces pueden encontrarse a principios de los 80, la amplia difusión que tiene hoy en día la Bioinformática, se debe principalmente al proyecto de secuenciación del genoma humano. Este proyecto comenzó en los 90 y es coordinado por el Depto de Energía y los Institutos Nacionales de Salud de los Estados Unidos. En principio se planteó una duración de 15

años, pero el aumento de la potencia computacional y el descenso de costos de laboratorio ha provocado que hoy en día ya se disponga de un primer borrador del genoma [38, 37].

La gama de problemas que abarca la bioinformática es muy amplia y como ejemplo podemos citar: la construcción de arboles filogenéticos para detectar antecesores comunes, el alineamiento simple y múltiple de secuencias, la construcción de mapas de genomas, la predicción de estructuras de proteínas, la comparación de moléculas, el agrupamiento y clasificación de estructuras proteicas, análisis de perfiles de expresión génica, y un largo etcétera. Dado que es imposible abarcarlos a todos, en esta memoria nos centraremos solamente en dos de ellos que, como veremos, tienen especial importancia: el problema de predicción de estructura en modelos basados en retículos, y el problema de emparejamiento estructural de moléculas.

Por lo tanto, con el objetivo de evaluar el comportamiento de *FANS* sobre estos problemas, el capítulo se encuentra organizado de la siguiente manera: en la sección 4.1 se presentan algunos conceptos muy básicos de biología, se describe brevemente el proyecto genoma humano y luego se analiza qué es la Bioinformática. Posteriormente en la sección 4.2 se define el problema de predicción de estructuras y se presentan los modelos basados en reticulados. La sección 4.3 esta dedicada a introducir el problema de comparación de estructuras y a revisar algunos trabajos relevantes del área. Con dicha sección se cierra una primera parte que podríamos denominar introductoria y comienza una segunda parte de aplicaciones donde se analiza la aplicación de *FANS* a los problemas descritos previamente. En la sección 4.4 se muestra la aplicación de *FANS* al problema de predicción de estructura en dos aspectos: primero, para analizar la influencia que tiene la codificación de las soluciones en los resultados obtenidos; y segundo, para comparar *FANS* frente a un *AG* y verificar una hipótesis respecto a la posibilidad de evitar el uso de una población de soluciones.

En la sección 4.5 se muestra la aplicación de *FANS* en el problema de emparejamiento estructural de moléculas. Finalmente, la sección 4.6 se dedica a la presentación de las conclusiones.

4.1 Conceptos Básicos

Las células son las unidades fundamentales de cualquier ser vivo y todas las instrucciones necesarias para dirigir sus actividades estan contenidas en la secuencia de ADN[1]. El ADN (ácido desoxyribonucleico) de todos los organismos está compuesto por los mismos componentes físicos y químicos, denominados bases, que se ordenan lado a lado en una estructura de doble hélice. El orden de estas bases contiene las instrucciones para crear un organismo con todas sus particularidades.

El genoma de un organismo está formado por el conjunto de moléculas de ADN, y el tamaño del mismo puede variar desde 600000 pares de bases en una bacteria, hasta los 3 billones que contienen los genomas humano y de ratón. Salvo algunas excepciones, todas las células humanas contienen una copia del genoma completo.

El ADN en el genoma humano está organizado en 46 cromosomas. Cada uno de ellos es una molécula cuya longitud se encuentra entre los 50 y 250 millones de pares de bases. Cada cromosoma contiene varios genes: las unidades básicas funcionales de la herencia y cada gen es "simplemente" una secuencia específica de bases que contiene las instrucciones para construir una proteína.

Hoy se sabe que los genes comprenden solamente el 2% del genoma humano; el resto contiene regiones no codificantes cuya función puede incluir la provisión de integridad estructural del cromosoma, la regulación de donde, cuando y en qué cantidad se fabrican las proteinas, etc.

Se estima que el genoma humano contiene entre 30000 y 40000 genes y aunque los genes atraen mucho la atención, en realidad son las proteínas las que realizan la mayor parte de las funciones de la vida y generan la mayoría de las estructuras celulares. Las proteínas son moléculas complejas, formadas por subunidades más simples denominadas aminoácidos, de los cuales existen 20 diferentes. La secuencia de aminoácidos y Las características químicas de los mismos causan que la proteína se pliegue en una estructura tridimensional

[1]Parte del material de esta sección está basado en [44] y en las páginas del Dr. Roderic Guigó Serra de la Univ Pompeu Fabra `http://www1.imim.es/~rguigo`. El lector interesado en profundizar sobre estos conceptos puede referirse también a [118]

que define su funcionalidad en la célula.

El conjunto de todas las proteínas de una célula se denomina proteoma. En constraste con el carácter estático del genoma, el proteoma cambia momento a momento en respuesta a miles de señales intra y extra celulares. La química de una proteína y su comportamiento está especificada por la secuencia de un gen, pero tambien por el número y la identidad de otras proteínas fabricadas en la célula al mismo tiempo y con las cuales ésta se asocia y reacciona.

La proteómica, definida como el área que estudia la estructura de las proteínas y sus actividades y relaciones, será objeto de investigación durante muchas décadas y ayudará a elucidar las bases moleculares de la salud y la enfermedad.

El proyecto de secuenciación del genoma humano (determinar la secuencia completa de bases del ADN) comenzó en los 90 y es coordinado por el Depto de Energía y los Institutos Nacionales de Salud de los Estados Unidos. En principio se planteó una duración de 15 años pero hoy en día ya se dispone de un primer borrador del genoma [38, 37]. Los objetivos del proyecto son: identificar los aproximadamente 30000 genes que existen en el ADN humano, determinar las secuencias de los 3 billones de pares de bases que constituyen el genoma humano, almacenar esta información en base de datos, mejorar las herramientas para el análisis de estos datos y considerar los aspectos éticos, legales y sociales involucrados en el proyecto. La dirección en Internet del proyecto es `http://www.ornl.gov/hgmis/` y una lista de recursos asociados se muestra en la Tabla 4.1.

Una consecuencia inmediata de los proyectos de secuenciación de genomas es la obtención, casi automática, de cantidades inmensas de datos y de una magnitud insólita en la historia de la biología. En este sentido, con la genómica, la biología se ha convertido en una ciencia de la información, tanto en la obtención de los datos genómicos primarios, como en su almacenamiento, análisis e integración.

La magnitud de la información que genera la investigación genómica es tal que, probablemente, supera la magnitud de la información que genera la investigación en otras disciplinas científicas. Por ejemplo, la base de datos de

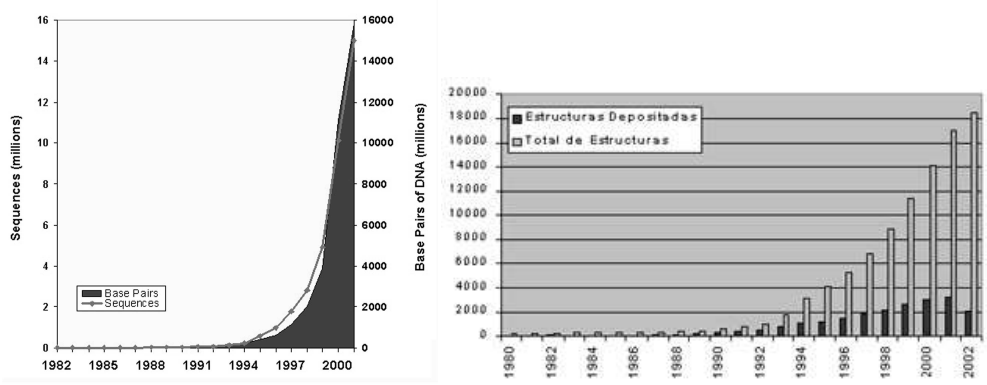
Información del Proyecto Genoma	
	`www.ornl.gov/hgmis`
Medicina y la Nueva Genética	
	`www.ornl.gov/hgmis/medicine/medicine.html`
Aspectos Eticos, Legales, y Sociales	
	`www.ornl.gov/hgmis/elsi/elsi.html`
Genomas para la Vida	
	`DOEGenomesToLife.org`
Borrador de la Secuencia del Genoma	
	`www.ornl.gov/hgmis/project/journals`
Galería de Imágenes	
	`www.ornl.gov/hgmis/education/images.html`
Recursos para Educadores	
	`www.ornl.gov/hgmis/education/education.html`
Recursos para Estudiantes	
	`www.ornl.gov/hgmis/education/students.html`
Carreras en Genómica	
	`www.ornl.gov/hgmis/education/careers.html`

Tabla 4.1: *Lista de Recursos sobre el Proyecto Genoma*

secuencias de ADN, GenBank[2], almacenaba en junio de 2002, 20.649.000.000 bases correspondientes a 17.471.000 sequencias. La base de datos de estructuras de proteínas, PDB[3], contiene al momento de escribir esta memoria, 18188 estructuras. El ritmo de crecimiento de ambas bases de datos se muestra en la Fig. 4.1.

En estos momentos, los ordenadores no clasificados para uso civil más potentes del mundo (en Celera Genomics y en Oak Ridge National Laboratory, por ejemplo, con una capacidad de cálculo cercana a los 2 Teraflops, billones de operaciones por segundo) se encuentran ya dedicados a la investigación biológica, concretamente a la obtención y al análisis de las secuencias de nucleótidos de los genomas conocidos; IBM, por su parte, anuncia en un plazo de cinco años un ordenador 500 veces más potente que Deep Blue, el orde-

[2] `www.ncbi.nlm.nih.gov/Genbank/index.html`
[3] `www.rcsb.org/pdb/`



(a) (b)

Figura 4.1: *Ritmo de crecimiento de GenBank (a) y PDB (b).*

nador que en mayo de 1997 derrotó a Kasparov. Su nombre es Blue Gene [4] y su objetivo, deducir tras un año de cálculo, la conformación tridimensional de una pequea proteína (de entre las decenas de miles codificadas en nuestro genoma) a partir de su secuencia de aminoácidos.

En este contexto, surge la Bioinformática como un área en la frontera entre la biología y las ciencias de la computación cuyo principal objetivo es el desarrollo y uso de técnicas matemáticas y computacionales para ayudar en el tratamiento masivo de datos y en la resolución de problemas de la biología molecular. Una definición más formal que aparece en [85] es:

> **Bio-Informática***: la bioinformática es una conceptualización de la biología en términos de moléculas (en el sentido de química física) y la aplicación de técnicas informáticas (derivadas de disciplinas como matemática aplicada, estadística, ciencia de la computación) para entender y organizar la información asociada con dichas moléculas en gran escala. En breve, bioinformática es un sistema de manejo de información para la biología molecular que tiene un gran número de aplicaciones prácticas.*

[4] `http://www.research.ibm.com/bluegene`

Esta área es especialmente atractiva puesto que las modelizaciones asociadas a gran parte de los problemas de biología que se utilizan, resultan ser NP-Completos y entonces, deben ser abordados mediante técnicas heurísticas. Por lo tanto, la resolución de estos problemas y la obtención de nuevos mecanismos de solución tienen y tendrán impacto, tanto en Informática como en Biología.

La gama de problemas que abarca la bioinformática es muy amplia y como ejemplos, podemos citar: la construcción de arboles filogenéticos para detectar antecesores comunes, el alineamiento simple y múltiple de secuencias, construcción de mapas de genomas, la predicción de estructuras de proteínas, la comparación de moléculas, el agrupamiento y clasificación de estructuras proteicas, el análisis de perfiles de expresión génica, y un largo etcétera.

Según Meidanis y Setubal[89], un algoritmo para un problema de biología molecular es un objeto que intenta servir a dos personas: el biólogo molecular, que pretende que el algoritmo sea *relevante*, es decir que resuelva el problema con todos los errores e incertidumbres que aparecen en la práctica; y el informático, que desea probar que el algoritmo resuelve eficientemente un problema bien definido y que está dispuesto a sacrificar relevancia por eficiencia. El equilibrio solo puede provenir de una interacción constante, que no es simple, pero que merece la pena. En la misma línea, se puede argumentar[5]

> " los biólogos querrán que los informáticos les suministren soluciones a sus problemas de gestión de datos, los matemáticos y expertos en computación andarán detrás de problemas intelectualmente llamativos, y los ingenieros pedirán a los dos grupos anteriores que les suministren especificaciones bien concretadas para que ellos puedan desarrollar su trabajo. Los distintos expertos habrán de acostumbrarse a emplear vocabularios y lenguajes comunes y a entender (sin minusvalorar) los problemas de los demás."

Algunos problemas importantes donde los enfoques basados en IA resultan prometedores incluyen la predicción y comparación de estructura de pro-

[5]Cita extraída de las páginas del Dr Enrique Iañez Pareja, del Depto. de Microbiología e Instituto de Biotecnología de la Univ. de Granada. `http://www.ugr.es/~eianez`

teínas, el diseño semi automático de drogas, la interpretación de secuencias de nucleótidos y la adquisición de conocimiento de los datos genéticos.

Una de los procedimientos básicos en el área de la Bioinformática, consiste en la búsqueda de semejanzas entre un fragmento de ADN recién secuenciado y los segmentos ya disponibles almacenados en grandes bases de datos como GenBank (`www.ncbi.nlm.nih.gov`). El hallazgo de emparejamientos aproximados permite predecir el tipo de proteína que especificará tal secuencia y esto no solo proporciona pistas sobre dianas farmacológicas prometedoras en las etapas iniciales de desarrollo de un medicamento, sino también permite eliminar alguna de ellas.

El problema de análisis, comparación y alineamiento de secuencias puede considerarse resuelto. Mejor dicho, hoy en día se dispone de algoritmos para resolver estos problemas razonablemente bien considerando que muchos de los problemas derivados resultan ser NP-Completos. Por lo tanto, estos problemas no serán objeto de este capítulo. El lector interesado en profundizar sobre problemas basados en secuencias, puede referirse a los trabajos de Meidanis y Setubal [89] y Gusfield [52]. Una visión general sobre los problemas del área también puede encontrarse en [57].

Los dos métodos clásicos para la búsqueda de secuencias similares en bases de datos son BLAST (Basic Local Alignment Search Tool) [3] y FAST [97]. Ambas referencias pertenecen a la presentación original de los métodos. Existen actualmente versiones mejoradas cuya descripción básica aparece también en [89]. Una revisión de métodos para el problema de alineamiento múltiple de secuencias aparece en [110, 25].

En las secciones siguientes se describen los dos problemas que abordaremos en esta memoria: el problema de predicción de estructura, y el problema de emparejamiento ("matching") estructural de moléculas.

4.2 El Problema de Predicción de Estructura

Las proteínas pueden considerarse los "arquitectos de la vida". Fueron descubiertas en 1838 y hoy se sabe que son los ingredientes principales de las células y que suponen más del 50% del peso seco de los animales. Todas las proteínas,

desde las humanas hasta las que forman las bacterias unicelulares, son el resultado de distintas combinaciones entre 20 aminoácidos que se enlazan entre sí mediante enlaces peptídicos formando una secuencia o cadena. Esta cadena adopta en el espacio una estructura tridimensional, la cual determina la funcionalidad biológica de la proteína; es una estructura con cavidades y salientes que permite el acoplamiento de otras proteínas para formar estructuras más complejas, o para bloquear el funcionamiento de otras.

Estructuralmente, las proteínas pueden ser analizadas a diferentes escalas. Se denomina *estructura primaria* de una proteína, a la secuencia de aminoácidos que la componen. Estos aminoácidos se agrupan en estructuras llamadas $\alpha - helices$, $laminas - \beta$ y *loops*, las cuales reciben el nombre de *estructuras secundarias*. Estas subestructuras se pliegan o "doblan" en el espacio tridimensional hasta alcanzar una configuración que se conoce como *estructura terciaria* o *estado nativo*. Es esta estructura tridimensional "final" la que determina la funcionalidad biológica de la proteína [21].

Es aceptado que uno de los elementos que más influye en la determinación de esta estructura, es el llamado *efecto hidrofóbico*. Los aminoácidos pueden clasificarse en hidrofílicos o hidrofóbicos según sea su comportamiento en un medio acuoso (es decir, si se "sienten a gusto" o no en el agua). En el proceso de plegado los aminoácidos hidrofóbicos tienden a agruparse en el centro de la molécula formando una especie de coraza interna, mientras que los hidrofílicos tienden a quedar expuestos al solvente.

La pregunta que define al *Problema de Predicción de Estructura* es: *dada la secuencia lineal de aminoácidos, cuál es la estructura tridimensional correspondiente?*

En ocasiones este problema recibe el nombre de Problema de Plegamiento de Proteína, o *Protein Folding Problem*, pero existe una diferencia con el problema de predicción de estructura. El primer problema tiene que ver con una simulación dinámica del proceso por el cual, una secuencia se pliega hasta alcanzar su estado nativo (estructura tridimensional). El segundo problema trata solamente con las estructuras finales sin involucrarse con las intermedias. Aunque los científicos intercambian ambos conceptos libremente, el lector debe

tener en cuenta que los algoritmos que se utilicen en este trabajo para este problema estarán dedicados a la predicción de estructuras y no a la simulación dinámica de plegamientos.

Dicho esto, vale la pena recalcar que la solución a este problema no es un tema menor; a pesar del gran desarrollo de las ciencias involucradas, químicos, biólogos, matemáticos y físicos no han podido establecer fehacientemente como la Naturaleza realiza este proceso en forma tan veloz y eficaz. Todos los trabajos relacionados con este tema asumen que la secuencia de aminoácidos es suficiente para determinar, completa y unívocamente, la estructura tridimensional. Esta hipótesis está relacionada con el siguiente experimento realizado por Anfinsen en 1961 [5]: trabajando *in vitro*, y modificando ciertas condiciones, se logra que una proteína se "desnaturalice" o desdoble (es decir, pierde su funcionalidad). Al restablecer las condiciones, la proteína retoma su forma tridimensional original muy rápidamente. Este experimento dió pie a la *Hipótesis Termodinámica* para explicar el proceso. Se asume que las proteínas, como sistema biológico, tienden a estabilizarse utilizando el menor esfuerzo posible. Esta conformación estable o de "mínima energía libre"se denomina estado nativo o estado funcional. Bajo esta hipótesis, el problema se puede plantear en términos de la minimización de alguna función de energía adecuada sujeta a restricciones.

4.2.1 Problemas en la determinación de la estructura

La determinación de la secuencia de aminoácidos que componen una proteína se realiza a partir del conocimiento de la secuencia de ADN que la codifica. El proceso de obtener la secuencia de nucleótidos que forman una cadena de ADN se denomina *secuenciación*. Desde el punto de vista tecnológico, la secuenciación puede considerarse un problema resuelto: por ejemplo, como ya dijimos, hoy en día se dispone de un borrador de la secuencia completa del genoma humano.

Para determinar la estructura terciaria de una proteína dada se pueden seguir dos caminos: utilizar métodos experimentales o trabajar en base a similaridades con proteínas conocidas.

Dentro de los métodos experimentales, se encuentran la cristalografía de rayos X y la resonancia magnética nuclear(NMR). El primero de ellos, permite obtener abundante información estructural pero la determinación de las condiciones de cristalización es muy complicada. Además el proceso de cristalización puede producir deformaciones en la estructura, por lo cual, la información debe ser analizada muy cuidadosamente. La NMR permite analizar las proteínas en solución pero provee información solamente sobre algunos tipos de átomos. Además se pierde detalle estructural y se deben asumir elementos como la geometría del esqueleto o *backbone* de la proteína.

Como generalmente ocurre con este tipo de métodos, son caros, las condiciones de experimentación deben ser cuidadosamente establecidas y no siempre es posible aplicarlos. Como consecuencia de esto, hoy en día existe una importante diferencia entre la cantidad de información disponible de secuencias de proteínas y de sus respectivas configuraciones espaciales.

La otra forma de determinar la estructura (o en última instancia, la función de una proteína), u obtener algunos indicios de la misma, es utilizar técnicas de modelización comparativas o por homología. Dada una secuencia de aminoácidos, se utilizan secuencias "similares" con estructuras conocidas, para determinar la estructura asociada. Este enfoque es posible ya que un pequeño cambio en la secuencia resulta, usualmente, en un cambio reducido en la estructura tridimensional.

Estos métodos constan basicamente de 3 pasos. En primer término, se realiza una búsqueda en bases de datos de secuencias para obtener cadenas "similares" con estructura conocida. Luego se realiza un alineamiento de las secuencias y se determinan las estructuras que serán utilizadas como modelos. Finalmente, se evalúa la secuencia sobre los modelos de acuerdo a varios criterios hasta que alguno de ellos brinde resultados satisfactorios. Naturalmente, si para una proteína no existen secuencias similares, este método no puede ser utilizado para determinar su estructura.

4.2.2 Modelizaciones del Problema

En las secciones previas, hemos explicado que los métodos experimentales (resonancia magnética nuclear, cristalografía de rayos X) utilizados para obtener la estructura tridimensional de una proteína, o para indagar en la dinámica del proceso de plegamiento, son muy costosos y además, poco útiles para secuencias de tamaño importante. Es por eso que el desarrollo de modelos adecuados para realizar simulaciones por computadora, es de gran importancia. Actualmente, con la tecnología computacional disponible, es imposible realizar simulaciones que involucren todas las interacciones proteína-solvente a nivel atómico.

En [35] E. Shakhnovich plantea que las proteínas representan sistemas muy complejos para permitir una modelización exacta y sugiere la necesidad de realizar simplificaciones en la formulación de los modelos de trabajo. Es decir, el desarrollo de técnicas computacionales que permitan indagar tanto en las propiedades cualitativas, como cuantitativas, de las proteínas debe ser realizado sobre modelos reducidos.

Un modelo para *PSP* es relevante si refleja alguna de las propiedades del proceso de formación de estructura en el sistema real. Una propiedad obvia es la de *Equivalencia Visual* entre la estructura predecida por el modelo y la estructura real. Otra propiedad es la de *Equivalencia de Comportamiento* entre el modelo y el sistema real. De acuerdo con la hipótesis termodinámica, el estado nativo de una proteína se corresponde con el estado de mínima energía libre y por eso los modelos basados en energía especifican una función de "costo" que asigna un valor de energía libre a cada estructura válida. Se asume que la estructura terciaria de la proteína se corresponderá entonces, con aquella conformación que minimice la función de energía.

Dentro de este tipo de modelos basados en energía se destacan los modelos basados en retículos. En ellos cada vértice de retículo es ocupado por un aminoácido de la cadena y aminoácidos consecutivos en la secuencia se ubican en posiciones adyacentes del retículo.

Entre otras características que los hacen atractivos, vale la pena destacar que permiten discretizar el espacio de conformaciones, facilitan el diseño y

prueba de variantes (por ej. respecto a la dimensionalidad, y la cantidad de elementos involucrados en la modelización), y pueden utilizarse como soporte para recolectar información estadística. En general, en cualquier modelo (reducido o no) deben estar claramente definidos cuatro elementos:

1. Cuáles son los aminoacidos a considerar.
2. Cómo se forman las secuencias válidas.
3. Cómo se representa un "plegado".
4. Cómo se mide la "bondad" de una estructura terciaria particular ("función de energía").

El punto 1 se refiere a que no todos los modelos utilizan los 20 aminoácidos que ocurren en la Naturaleza; por ejemplo, en el modelo de Dill [34] que veremos más adelante, solo se utilizan 2. El segundo punto está relacionado con el hecho que las secuencias válidas sean más restringidas que cualquier concatenación de aminoácidos. Por ejemplo, se pueden establecer restricciones respecto a la cantidad de aminoácidos del mismo tipo que pueden aparecer consecutivos. Para representar las estructuras o "plegados", se pueden utilizar [103]:

- *Coordenadas Cartesianas*: cada aminoácido se representa con 2 o 3 coordenadas, dependiendo si la estructura pertenece al plano o al espacio tridimensional.
- *Coordenadas Internas*: la posición de cada aminoácido se define en términos de sus vecinos, especificando distancias, ángulos, etc.
- *Matriz de Distancias*: describe la estructura en términos de una matriz que contiene las distancias para cada par de aminoácidos.

Los modelos en retículos utilizan las coordenadas internas para modelizar las estructuras terciarias de las proteínas. Esto es, fijada la posición del aminoácido i, existen δ valores para representar la posición del $i+1$ dependiendo del retículo utilizado. Por ejemplo, supongamos un retículo cuadrado donde

ya se han ubicado el primer y segundo aminoacidos. Para el tercer aminoácido, existirán $\delta = 3$ posiciones posibles las cuales provienen del conjunto posible de direcciones *Arriba, Abajo, Izquierda, Derecha* que indican la posición respecto al predecesor en la cadena. Bajo este modelo una estructuras queda, entonces, representada por una cadena $s \in \{Arriba, Abajo, Izquierda, Derecha\}^+$. Si δ es un valor pequeño, todas las combinaciones posibles pueden ser testeadas. Potencialmente, existen δ^N posibles estructuras asociadas a una secuencia de longitud N y *PSP* en estos modelos implica encontrar la única, o unicas, "correctas".

El modelo más simple de *PSP* en retículos, es el llamado **modelo de Dill** [34]. Solo considera 2 tipos de aminoácidos, donde cada tipo representa si es hidrofóbico (representado con una H) o hidrofílico (representado con una P). Una proteína entonces, se modeliza como secuencia $w \in \{H, P\}^+$. Graficamente consideramos a los aminoácidos H de color negro y a los P de color blanco.

Generalmente se considera que la proteína se encuentra inmersa en un retículo cuadrado o cúbico (según sean 2 o 3 dimensiones), donde cada posición es ocupada por a lo sumo un aminoácido. La correspondencia entre aminoácidos y posiciones se llama *embedding*, y cuando esta es inyectiva se denomina *self avoiding*, es decir, no existen dos aminoacidos que ocupen la misma posición (la estructura no tiene cruces).

La función de energía utilizada, solo tiene en cuenta las interacciones entre aminoácidos que sean adyacentes en el retículo, pero no consecutivos en la secuencia (*vecinos topológicos*). Cada interacción de este tipo se denomina *bond* o contacto. Dada una secuencia con n aminoácidos , $S = (s_1, s_2, \ldots, s_n)$, con $s_i \in \{H, P\}$, un plegado para S, $fold(S) = X = (x_1, x_2, \ldots, x_n)$ dispuesto en un retículo $\mathcal{L}$, y una matriz de interacción $\varepsilon(s_i, s_j)$, una función de energía posible es:

$$E(S, X) = \sum_{i}^{n} \sum_{j>i+1}^{n} \varepsilon_{i,j} * \Delta(x_i, x_j) \tag{4.1}$$

donde $\Delta(x_i, x_j) = 1$ si x_i y x_j son adyacentes en el retículo y no consecutivos en la cadena y 0 en caso contrario. El término $\varepsilon_{i,j} = \varepsilon(s_i, s_j)$ es el valor de la

	H	P
H	-1	0
P	0	0

	H	P
H	-3	-1
P	-1	0

Tabla 4.2: *Matrices de interacción* $\varepsilon_{i,j}$

fila i, columna j en la matriz de interacción ε. En la Tabla 4.2.2 se muestran dos matrices de interacción posibles.

Con estos elementos se establece que *Resolver* PSP *en este modelo es equivalente a minimizar esta función de energía.* De forma equivalente, se puede plantear como objetivo maximizar el número de contactos.

Las estructuras en estos modelos se representan utilizando el sistema de coordenadas internas, de las cuales aparecen dos variaciones: la codificación mediante coordenadas absolutas o coordenadas relativas. Cuando se utiliza la codificación absoluta, las estructuras se representan mediante una lista de movimientos absolutos en el espacio corrrespondiente. Por ejemplo, si el soporte es un retículo cuadrado bidimensional, entonces una estructura s se codifica como un string $s = \{Arriba,\ Abajo,\ Izquierda,\ Derecha\}^+$ o, a partir de ahora, $s = \{U, D, L, R\}^+$. Bajo la codificación relativa, cada movimiento debe interpretarse en términos del anterior, de una forma que recuerda a los movimientos de la tortuga del lenguaje LOGO. En este caso, la estructura es un string $s = \{Avanzar, GirarIzquierda, GirarDerecha\}^+$. En forma breve, $s = \{F, L, R\}^+$. En la Fig. 4.2 se muestra una instancia de este modelo en 2D. La secuencia es $P = HPPHPHHPHHPH$ y la estructura se puede codificar como $s = RURDRDLLDLU$ (utilizando codificación absoluta) o $s = FLRRLRRFLRR$ (codificación relativa). Como matriz de interacción para medir la energía de esa conformación utilizamos la primera matriz de las mostradas en la Tabla 4.2.2. La estructura tiene 4 "contactos", los cuales se indican con líneas de puntos.

A pesar de la sencillez del modelo, es posible capturar uno de los elementos fundamentales del proceso de plegado: las interacciones hidrofóbicas. Es interesante ver en la Fig. 4.2 como aparecen las características de hidrofo-

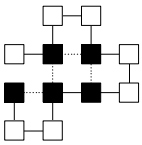


Figura 4.2: *Instancia del modelo HP en retículo cuadrado. Bonds = 4*

(a) (b)

Figura 4.3: *retículo triangular en 2 y 3 dimensiones.*

bicidad de los aminoácidos. Notese que, de alguna manera, los aminoácidos hidrofóbicos están "aislados del exterior" por una barrera de aminoácidos hidrofílicos que los rodean.

La utilización de retículos cuadrados o cúbicos como soporte para discretizar el espacio de conformaciones, trae aparejado el problema de *restricción de paridad* (o *parity constraint*). Esta restricción esta asociada al hecho que en esos retículos, los aminoácidos que se encuentren a distancia impar nunca pueden ser vecinos topológicos. Como resultado, secuencias del tipo $(HP)^+$ no pueden formar ningún bond, aunque en un "espacio real" sí podrían hacerlo. Este problema se soluciona si se utilizan, por ejemplo, retículos triangulares. En ellos, para cualquier par de elementos $(x,\ y)$ no consecutivos de la secuencia, es posible encontrar un *embedding* en el retículo tal que x sea vecino topológico de y. La Fig. 4.3 muestra instancias del modelo HP dispuestas en el retículo triangular en 2 y 3 dimensiones. La estructura en dos dimensiones tiene un valor de $bonds = 6$.

Se han realizado varios intentos para tratar de establecer la complejidad computacional del *PSP* sobre estos modelos. Sin embargo, cada demostración

involucra una definición un poco "diferente" de lo que significa *PSP* (en el caso más simple, *PSPes* maximizar el número de bonds), por lo que al momento de establecer comparaciones entre modelos se debe tener en cuenta si el problema definido es el mismo.

En 1992, Ngo and Marks [92] muestran que un modelo tridimensional de predicción de estructura es *NP-Hard* reduciendo desde el problema de partición (*partition problem*). En 1993, Fraenkel [40] muestra que un modelo general en dos dimensiones (*MEP*) es *NP-Hard* a partir de una reducción desde el problema de *three dimensional matching (3DM)*. En el mismo año, Unger y Moult [111] utilizan un modelo similar al de Dill en tres dimensiones y prueban su *NP-Completitud* reduciendo desde el problema de *optimal linear arrangement* [42]. En 1997, Paterson y Przytycka [95] muestran que *String Folding* es *NP-Completo* en Z^2 y Z^3.

Por la misma época aparecen algoritmos de aproximación. Hart e Istrail [65] presentan un algoritmo para retículos cuadrados en dos y tres dimensiones con factores de aprox. de 3/8 para tres dimensiones y de 1/4 para dos. Simultaneamente, Farach et. al. [2] presentan conjuntos de reglas para realizar "foldings" en retículos triangulares (modelo de Dill) y logran obtener factores de aproximación para cada conjunto.

Ya a principios de 1998, Crescenzi et. al. [30] probaron que *PSP* en el modelo HP en dos dimensiones es *NP-Completo* y Berger y Leighton [16] lo mostraron para el retículo cúbico. Según nuestro conocimiento, la última referencia en el tema pertenece a Atkins y Hart [6] quienes demuestran que una versión en retículos 3d del problema cuyas instancias contienen hasta 12 tipos de aminoacidos es NP-Hard.

4.3 El Problema de Comparación de Estructuras

A medida que el número de estructuras proteicas conocidas aumenta, la necesidad de disponer de algoritmos para analizar dichas estructuras tridimensionales también se incrementa. Por ejemplo, la búsqueda de subestructuras comunes en un conjunto resulta de interés para revelar relaciones entre diferentes proteínas, para inferir similaridades en la función y para descubrir

origenes evolutivos comunes. En la actualidad, existe un consenso respecto a que las similaridades en proteínas distantes se preserva más a nivel de estructura tridimensional, aun cuando practicamente no exista relación a nivel de secuencia de aminoácidos.

Desde el punto de vista algorítmico, la comparación de estructuras tridimensionales de proteínas es un problema muy complejo. La búsqueda de técnicas computacionales que permitan resolverlo (aunque sea en forma aproximada), está justificada porque dichas herramientas pueden ayudar a los científicos en el desarrollo de protocolos para el diseño de nuevas drogas, la identificación de nuevos tipos de estructuras proteicas, la organización del conjunto de proteínas conocidas, etc.[60, 72].

4.3.1 Consideraciones Generales

La forma más natural de comparar dos objetos, cada uno representado por una colección de elementos, consiste en tratar de encontrar una correspondencia entre estos elementos. Siendo A y B dos objetos con elementos $a_1, a_2, \ldots, a_m$ y $b_1, b_2, \ldots, b_n$ respectivamente, se define una *equivalencia* como un conjunto de pares $E(A,B) = (a_{i_1}, b_{j_1}), (a_{i_2}, b_{j_2}), \ldots, (a_{i_r}, b_{j_r})$. Esta equivalencia es también un *alineamiento* si los elementos de A y B están ordenados y si los pares en $E(A,B)$ son colineales, es decir, si $i_1 < i_2 < \ldots < i_r$ y $j_1 < j_2 < \ldots < j_r$.

En la actualidad existen varios algoritmos que, a partir de dos estructuras y una función de costo, permiten obtener la equivalencia de mejor costo. Sin embargo, el problema general es NP-Hard y por lo tanto, se deben realizar simplificaciones en la búsqueda o en la formulación de la función de costo para poder abordarlo.

Una forma de realizar la comparación de estructuras es mediante la superposición rígida de ambos objetos. La superposición óptima se puede determinar exactamente y requiere: a) un vector de translación para mover una de las estructuras sobre el sistema de coordenadas de la otra, y b) una matriz de rotación para emparejar ambos objetos. La calidad de la superposición se evalúa en términos del RMSD (Root Mean Square Deviation) de *coordenadas* cuya expresión es:

$$RMSD_c = \sqrt{\frac{1}{r}\sum_{i=1}^{r}(a_i - b_i)^2} \tag{4.2}$$

donde $a_i \in A, b_i \in B$ se entienden como el conjunto de coordenadas asociadas al objeto i. Por ejemplo, cuando se trata de proteínas, cada objeto contiene las coordenadas del carbono central del residuo.

Otra forma de evaluar las equivalencias, es utilizar el RMSD de las *distancias* que evita la necesidad de determinar una rotación y una traslación. La expresión correspondiente es:

$$RMSD_d = \frac{1}{r}\sqrt{\sum_{i=1}^{r}\sum_{i=1}^{r}(d_{ij}^A - d_{ij}^B)^2} \tag{4.3}$$

donde cada d_{ij}^C es la distancia entre los elementos i y j en la estructura C.

Ambas medidas se utilizan para evaluar equivalencias y no alineamientos, y experimentalmente está demostrado que existe una relación lineal entre ambas. Una diferencia importante entre las medidas es que $RMSD_d$ es invariante bajo reflexiones, lo cual implica que si B es la imagen especular de A, entonces $RMSD_d(A, C) = RMSD_d(B, C)$ y $RMSD_d(A, B) = 0$.

En síntesis, el problema de comparar dos estructuras se formula usualmente como el problema de encontrar equivalencias con valores bajos de RMSD. Sin embargo, pueden existir diferentes soluciones (equivalencias) con valores similares, y decidir cual de ellas representa la "correcta" no es una tarea simple. Por otro lado, la utilización de medidas de similaridad estructural basadas en RMSD presenta un problema y es la sensibilidad frente a la presencia de puntos extremos ("outliers"). En consecuencia existen otras formas de evaluar la similaridad que no serán reseñadas aquí por no ser este el objeto central de esta tesis. El lector interesado puede referirse al trabajo de Kohel [72] y las referencias allí citadas, o al trabajo de May [88] donde se comparan hasta 37 medidas de similaridad.

En las secciones siguientes, y a modo de ejemplo, se describen algunos algoritmos que se utilizan actualmente para realizar la comparación estructural de proteínas que están basados en la comparación de matrices de distancia [59],

en la detección de cliques maximales [41] y en la superposición de mapas de contacto [82]. Otras referencias interesantes, no descriptas aquí, involucran la utilización de AG's [87, 109] y herramientas basadas en las técnicas de visión artificial [119, 83].

4.3.2 Comparación de Estructuras vía Matrices de Distancia

La utilización de matrices de distancia para la comparación de proteínas dió lugar a uno de los algoritmos más difundidos en la actualidad: DALI [59].

A partir de dos proteínas A y B, DALI calcula el valor de una equivalencia entre los residuos mediante una función S de la forma:

$$S = \sum_{i=1}^{L} \sum_{j=1}^{L} \phi(i,j) \tag{4.4}$$

donde i, j se refiere al par de residuos equivalentes, L es la cantidad de residuos en la equivalencia y ϕ es una medida de similaridad basada en las distancias entre los carbonos centrales de cada residuo. Para ϕ se utiliza una medida "elastica" ϕ^E, que es tolerante al efecto acumulativo de distorsiones geométricas graduales, y cuya definición es:

$$\phi^E(i,j) = \begin{cases} \theta^E - \frac{|d_{ij}^A - d_{ij}^B|}{d_{ij}^*} w(d_{ij}^*), & \text{si } i \neq j \\ \theta^E & \text{si } i = j \end{cases} \tag{4.5}$$

donde d_{ij}^* es el promedio entre d_{ij}^A y d_{ij}^B, $\theta^E = 0.2$ es un umbral de similaridad y $w(r) = exp^{(-r^2/400)}$ es una función que disminuye el peso de las distancias mas significativas.

DALI consta de dos pasos. Una vez construidas las matrices de distancia para ambas proteínas, el primero paso consiste en realizar una comparación sistemática entre todas las submatrices $(i_A \ldots i_A{+}5, j_A \ldots j_A{+}5)$ de la proteína A con todas las submatrices $(i_B \ldots i_B{+}5, j_B \ldots j_B{+}5)$ de la proteína B. Todos aquellos pares "similares", denominados patrones de contacto, se almacenan en una lista que constituirá el material para construir el alineamiento.

El objetivo del segundo paso es combinar los pares obtenidos en el paso 1, para dar lugar a pares de mayor longitud que maximicen la función de simila-

ridad. La construcción del alineamiento a partir de los patrones de contacto es un problema combinatorio complejo y se realiza mediante el método de Monte Carlo. El algoritmo también incluye mecanismos adicionales para refinar la lista de pares similares y los alineamientos obtenidos.

DALI se encuentra accesible en `http://www2.ebi.ac.uk/dali` desde donde se pueden ejecutar comparaciones estructurales. Además existe una versión de distribución *DaliLite* que permite ejecutar el algoritmo en forma local, y se puede obtener desde `http://jura.ebi.ac.uk:8765/~holm/DaliLite`

4.3.3 Comparación de Estructuras vía Cliques

La representación de moléculas en términos de grafos, permite identificar las relaciones estructurales entre pares de ellas mediante el uso de algoritmos que resuelvan el problema de *subgrafo común maximal*(MCS). En esta sección se describe el trabajo de Gardiner et.al. [41] donde se utiliza esta técnica para la detección de farmacóforos comunes en dos moléculas. Un enfoque común para la detección de farmacóforos comienza con la identificación de estructuras que sean activas en algún aspecto de interés. Poseteriormente se utiliza un algoritmo para resolver MCS para identificar la estructura común más grande entre las estructuras. Se asume que el patrón en común es el responsable, o está involucrado, en en el farmacóforo responsable de la respuesta biológica observada.

Los algoritmos para resolver MCS operan intentando identificar cliques en un *grafo de correspondencia*. Un clique es un subgrafo de un grafo en el cual, cada nodo está conectado con todos los demás y a su vez, no está contenido en ningún subgrafo mayor con dicha característica.

Dados dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, se define el grafo de correspondencia $C = (V, E)$ donde $V = \{(v_1, v_2)$ con $v_1 \in V_1,\ v_2 \in V_2$ y $\rho(v_1, v_2) = True\}$. El predicado $\rho(v_1, v_2)$ retorna $True$ si v_1 y v_2 son compatibles. Si el valor de las aristas entre $(v_i, v_j \in V_1)$ es el mismo que entre $(v_x, v_y \in V_2)$, entonces existirá en el grafo C una arista desde el vértice $(v_i \in V_1, v_x \in V_2)$ al $(v_j \in V_1, v_y \in V_2)$

Un resultado de teoría de grafos [14], muestra que los MCS's entre G_1 y

G_2 se corresponden con los cliques en el grafo de correspondencia. Es decir, se busca el conjunto de elementos(átomos) más grande cuyas distancias interatómicas sean similares (en términos de cierto valor de umbral definido por el usuario) en ambas moleculas.

En el trabajo reseñado, los vertices representan los elementos de estructura secundaria (α-hélices y láminas-β) y las aristas contienen los ángulos y distancias entre dichas estructuras.

Los autores compararon la eficiencia de 5 algoritmos de detección de cliques y concluyen que el más eficiente para detectar el clique máximo es el algoritmo de Carraghan y Pardalos [24]. Además, sugieren que el algoritmo clásico de Bron-Kerbosch [22] es la opción a elegir si se desean obtener todos los cliques maximales en lugar de unicamente el máximo. Finalmente, se indica que estas conclusiones son válidas para grafos de "baja" densidad [6] como los implicados por la representación utilizada.

4.3.4 Comparación de Estructuras vía Superposición de Mapas de Contacto

Un mapa de contacto puede verse como un grafo donde los nodos representan los residuos de las proteínas, y dos residuos i, j estarán conectados por una arista ("contacto") si la distancia entre ellos está por debajo de cierto umbral. Un alineamiento entre dos mapas de contacto especifica los residuos que se consideran equivalentes y el valor de una superposición de ambos mapas (CMO) está dado por el número de contactos (aristas) en el primer mapa cuyos extremos (los nodos que conecta) estén alineados con residuos del segundo mapa que también estén en contacto.

Planteado como problema de optimización, lo que se busca es la superposición máxima. Este problema es NP-Completo [50, 73] y en [82] se presenta el primer algoritmo riguroso para resolverlo.

El enfoque está basado en una modelización del problema mediante programación lineal entera y su resolución vía una estrategia de ramificación y corte "branch and cut" que utiliza heurísticas para construir cotas inferiores

[6] Siendo e el número de vértices y k la de aristas, la densidad se calcula como $2k/e(e-1)$

en los nodos de ramificación. El problema CMO se reduce al de encontrar el máximo conjunto independiente (MIS) en grafos especiales que contienen aproximadamente 10000 vertices para instancias de CMO de aprox. 300.

Un conjunto independiente es un conjunto de vértices entre los cuales no existe ninguna arista. Es equivalente al problema de encontrar cliques y hasta el momento no existen algoritmos que permitan resolver eficientemente instancias de unos pocos cientos nodos. Sin embargo, los autores consiguen resolver instancias más grandes debido a que el grafo subyacente es perfecto. Esta característica permite encontrar cliques en tiempo polinomial.

La estrategia de *branch and cut* utiliza la relajación lineal(PL) para obtener cotas superiores mientras que las cotas inferiores se obtienen mediante implementaciones de un algoritmo genético y una heurística de búsqueda local. Los "cortes" que se agregan permiten reducir el espacio de soluciones fraccionales sin eliminar ninguna solución entera factible, y por lo tanto el espacio de búsqueda se restringe.

La utilización de la PL como mecanismo de obtención de cotas superiores establece actualmente un límite al tamaño de las instancias: proteínas entre 64 y 72 residuos con 80 a 140 contactos. Se comprueba además que el tiempo utilizado para resolver cada LP varía entre 1 minuto y dos horas. A pesar de esta limitación, debemos resaltar que este trabajo es el único que garantiza la respuesta óptima lo cual es fundamental, por ejemplo para establecer comparaciones fiables entre métodos.

4.4 *FANS* para el Problema de Predicción de Estructura

4.4.1 Introducción

En esta sección nos centraremos en dos aspectos importantes relacionados con el modelo HP [34] del problema de predicción de estructura.

En primer lugar, y según nuestro conocimiento, no existió hasta hace poco tiempo una comparación directa entre la codificación relativa y la absoluta. El primer estudio se presentó en [76] y fue realizado en el marco de los algoritmos

evolutivos, donde se evaluaba el comportamiento de un algoritmo genético sobre ambas codificaciones para los retículos cuadrado, cúbico y triangular y se reportaban los resultados. Sin embargo, estos resultados nada dicen respecto a cual será el comportamiento en otros métodos. Es decir, no se pueden extrapolar directamente estos resultados a otros métodos heurísticos sin contar con mas evidencia.

Es por eso que en esta sección nos planteamos como primer objetivo analizar si la diferencia de resultados derivada de la utilización de ambas codificaciones tambien aparece en *FANS*. De esta manera, ganaremos evidencia acerca de la influencia de la codificación y podremos establecer guías y recomendaciones para futuras aplicaciones de *FANS* y otros métodos sobre este problema.

En segundo lugar, es bien sabido que desde un punto de vista práctico, los AG's son capaces de obtener buenos resultados sobre *PSP* [112, 96, 26, 77, 113, 69, 103, 105]. También resulta relevante el reciente survey de Greenwood at.al[51]. Sin embargo existe un problema "teórico" derivado del uso de operadores de cruce clásicos y una representación basada en coordenadas internas. En T. Jones[66] se hace una distinción entre la idea y la mecánica del operador de cruce. La *idea* de este operador es permitir el intercambio de información entre los individuos de la población; la *mecánica* del mismo es simplemente una implementación particular del operador.

Con esta idea en mente, en [77, 75] se discutió e implementó un AG para *PSP* y se pudo mostrar que efectivamente, el operador de cruce era incapaz de transferir información de padres a hijos. Es decir, la *idea* no estaba funcionando. La verificación de este hecho se realizó mediante el *"headless chicken test"* [66]. Este test compara los resultados de un AG que utiliza un cruce estandar frente a otro AG que utiliza un operador de cruce aleatorio. El cruce aleatorio toma un padre de la población actual mientras que el otro es generado aleatoriamente. Es claro este comportamiento refleja unicamente la mecánica y no la idea del cruce y provoca que el operador se comporte como una macromutación.

Los resultados del experimento mostraron que el AG con cruce aleatorio

fue capaz de obtener los mismos o incluso mejores resultados que el AG con cruce estandar. Por lo tanto, dado que no existe intercambio de información entre individuos, el uso de uno población de soluciones ya no es necesario: *potencialmente* los mismos resultados se podrían obtener mediante mutaciones sobre un único individuo.

El segundo objetivo de esta sección es mostrar como *FANS* es capaz de obtener tan buenos resultados como los de un AG, dando en consecuencia evidencia experimental a la hipotesis anterior y mostrando la potencia de nuestra heurística basada en conjuntos difusos.

Para alcanzar los dos objetivos planteados, se comienza con una descripción de algunos aspectos relevantes en el diseño de algoritmos para *PSP*. Luego se muestran los resultados sobre la influencia de la codificación en un algoritmo genético. A continuación se presentan las definiciones para los componentes de *FANS* utilizadas para tratar con el *PSP* y luego se muestran los experimentos y resultados realizados para comparar la codificación relativa frente a la absoluta. Posteriormente, se muestran los experimentos y resultados referidos a la comparación de *FANS* frente a un AG.

4.4.2 Diseño de Algoritmos para *PSP*

En esta sección se analizan tres factores algoritmicos que influyen sobre el diseño y performance de algoritmos para el *PSP*. Estos factores son [76]: la representación de las estructuras, la función de costo y el manejo de restricciones. Si bien el análisis se realiza sobre los algoritmos genéticos, los mismos conceptos son aplicables a otros métodos.

Codificación mediante Coordenadas Internas

Cuando se trabaja sobre modelos en retículos, es usual representar las estructuras mediante coordenadas internas. Sin embargo, no se han realizado estudios para verificar que representación es más efectiva: la absoluta o la relativa. En general, se elige alguna de ellas sin justificación previa [96, 112, 77].

En esta sección, y en el contexto de AG's, mostraremos como ambas codificaciones provocan comportamientos diferentes en los operadores genéticos

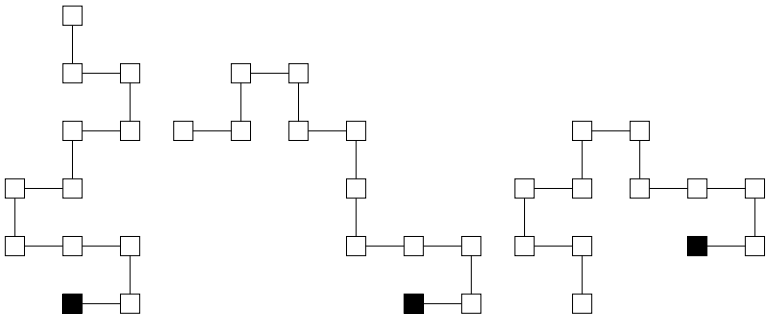


Figura 4.4: *En (b) se muestra el efecto de una mutación sobre la posición 6 de la estructura (a). El movimiento 'R' fue mutado a 'F' produciendo un efecto de "palanca" de 90 grados en sentido positivo. En (c) la 'R' fue mutada a 'L' produciendo un efecto de palanca de 180 grados en sentido positivo.*

estandar, lo cual afecta la capacidad de búsqueda del método. Como ejemplo, se utilizará el retículo cuadrado en dos dimensiones.

Mutación bajo Codificación Relativa: consideremos el efecto de una mutación de un punto sobre la estructura que se muestra en la Figura 4.4(a). La codificación relativa de dicha estructura es $S_{rel} = FLLFR\mathbf{R}LRLLR$ comenzando desde el aminoácido H. Una mutación en la posición 6, puede producir $S^1_{rel} = FLLFR\mathbf{F}LRLLR$ o $S^2_{rel} = FLLFR\mathbf{L}LRLLR$. La representación gráfica de estas estructuras se muestra en las Figuras 4.4(b) y 4.4(c) respectivamente.

Este ejemplo nos permite observar que una mutación en un punto bajo codificación relativa produce un efecto de rotación de la estructura en el punto de mutación. Para producir el mismo efecto bajo la codificación absoluta, es necesario realizar una macromutación; es decir, se deben modificar simultaneamente varias posiciones.

Es posible definir un operador de rotación en codificación absoluta de la siguiente manera: dado un punto donde se producirá la rotación, se cambiarán todas las posiciones siguientes de acuerdo a una transformación que dependerá del ángulo de rotación seleccionado. Por ejemplo, si se deseara una rotación

de 90 grados en sentido horario, entonces las transformaciones de movimientos serían: $U \mapsto R, D \mapsto L, R \mapsto D, L \mapsto U$.

En el ejemplo anterior, la estructura de la Figura 4.4(a) se codifica como $S_{abs} = RULLU\mathbf{RURULU}$ mientras que la primera estructura mutada, Figura 4.4(b)), es $S^1_{abs} = RULLU\mathbf{ULULDL}$. La tercera estructura, Figura 4.4(c), es $S^2_{abs} = RULLU\mathbf{LDLDRD}$.

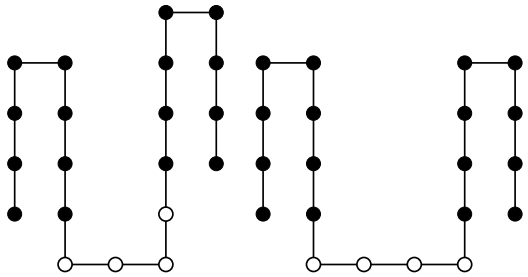
Mutación bajo Codificación Absoluta: una mutación de un punto bajo la codificación absoluta, mantiene la orientación de la estructura restante sin cambios. Para obtener el mismo efecto en la codificación relativa, es necesario modificar dos posiciones consecutivas.

Debe notarse que existen restricciones para transformar la mutación de un punto en codificación absoluta a la mutación en codificación relativa, ya que en el primer caso es factible obtener estructuras con movimientos UD o LR que generan colisiones no representables en la codificación relativa.

Formulación de la Función de Costo

La Figura 4.5 ilustra dos estructuras o conformaciones de una secuencia que está formada por dos dominios conectados por una cadena hidrofílica. Dado que el modelo HP básico solo "premia" los contactos entre aminoacidos hidrofóbicos, unicamente las subestructuras compactas aportarán la energía a dichas estructuras. Sin embargo, parece claro que la estructura de la Fig. 4.5(a) es más cercana a la conformación óptima que la estructura en Fig. 4.5(b).

Este tipo de disparidad entre el valor de energía y cuán compacta es la estructura, puede ser remediado extendiendo la función de energía de forma tal que permita un término dependiente de las distancias entre aminoacidos hidrofóbicos. Dado que las distancias entre aminoacidos forman un conjunto numerable, es posible construir un potencial basado en las distancias que preserve el orden de las conformaciones existentes en el modelo básico, logrando además, una distinción más fina para conformaciones con igual número de contactos hidrofóbicos. Por ejemplo, si d_{ij} es la distancia entre dos aminoacidos hidrofóbicos H_i y H_j, entonces se podría utilizar



(a) (b)

Figura 4.5: *Dos conformaciones con la misma energía en el modelo HP básico. En cambio, la estructura (a) tiene un valor de energía menor en el modelo HP modificado.*

$$\hat{E}_{H_iH_j}(d_{ij}) = \begin{cases} -1 & , d_{ij} = 1 \\ -1/(d_{ij}^k N_H) & , d_{ij} > 1 \end{cases} , \tag{4.6}$$

donde N_H es el número de hidrofóbicos en la secuencia y $k = 4$ para el retículo cuadrado ($k = 5$ para los retículos triangular y cúbico)[7].

Manejo de Restricciones

Dos clases de restricciones deben ser garantizadas para obtener una estructura factible: (1) la conectividad de la secuencia y (2) la ausencia de colisiones o cruces en la conformación.

Una de las motivaciones más fuertes para utilizar representación basada en coordenadas internas, es que la primera restricción se maneja implicitamente. Esto no ocurre si por ejemplo, se utilizan coordenadas cartesianas, en cuyo caso hay que manejar explicitamente esta restricción.

Para manejar la segunda clase de restricción cuando se utilizan coordenadas internas, existen dos posibilidades. Primero, restringir la búsqueda considerando unicamente las conformaciones sin colisiones. Sin embargo, este procedimiento no es adecuado en problemas como el *PSP* ya que el camino más corto desde una conformación factible a otra factible puede ser muy largo en

[7]Para un análisis más detallado de esta función, referirse a [73]

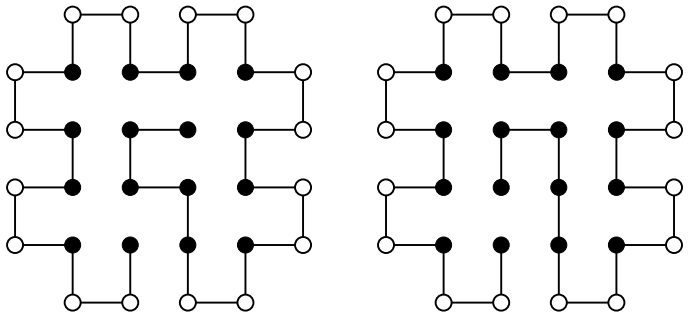


Figura 4.6: *Las dos conformaciones se encuentran "cercanas" si se admiten movimientos a través de conformaciones con colisiones.*

comparación con el camino más corto que se obtiene al pasar por conformaciones con colisiones. Por ejemplo, la estructura mostrada en la Figura 4.6(a) se puede transformar en la mostrada en 4.6(b) utilizando solamente 3 cambios. Estos cambios generan conformaciones intermedias con cruces, pero realizar la transformación utilizando movimientos entre conformaciones sin cruces sería mucho más costoso.

El segundo enfoque para manejar las colisiones, es utilizar términos de penalización en la función de costo que guien al algoritmo hacia conformaciones factibles. Se han utilizado dos métodos de penalización para *PSP* en el modelo HP. En el primero, se agrega una penalización para cada par de aminoácidos que se ubiquen en el mismo punto del retículo. Con este método pueden darse hasta $O(n^2)$ penalizaciones.

La otra opción es penalizar cada punto del retículo en el cual se ubiquen dos o más aminoacidos. De esta forma, existirán a lo sumo $O(n)$ penalizaciones. Patton [96] extienden este enfoque para evitar que los aminoacidos que provocan colisiones contribuyan a la función objetivo de forma positiva.

Cuando se evalúan estos metodos de penalización es importante considerar si son capaces o no de guiar la búsqueda hacia regiones factibles. Por ejemplo, si se utiliza un método de penalización que aplique una constante fija C por colisión, pueden existir problemas si no se incluye la extensión de Patton et.al.[96]. Para algunos valores de C es posible construir ejemplos don-

de el valor óptimo de energía con el método de penalización, sea diferente al correspondiente en el módelo HP básico.

También es importante analizar la eficacia de la penalización para comprender si facilitan o no la optimización. Por ejemplo, la formulación extendida presentada por Patton et.al. puede derivar en una búsqueda menos efectiva que otros métodos. Cuando se evita que los aminoacidos hidrofóbicos contribuyan a la función de costo porque provocan colisiones, el espacio de búsqueda puede tener regiones muy amplias donde muchas soluciones diferentes poseen el mismo valor de costo, y en consecuencia, pueden provocar que el problema de optimización tratado se vuelva mas difícil de resolver.

Estas consideraciones permiten recomendar el uso de un término de penalización fijo C por cada colisión que esté basado en la cantidad de aminoacidos hidrofóbicos N_H que aparecen en la secuencia. Por ejemplo, si en el retículo cuadrado se define $C = 2N_H + 2$, se consigue que cualquier conformación con colisiones tenga un valor de energía positivo mientras que las conformaciones factibles tendrán un valor negativo. De esta manera, la conformación óptima de una secuencia en el modelo HP es estrictamente mejor que la mejor conformación con penalizaciones.

4.4.3 Influencia de la Codificación en un AG

Ahora se describen los experimentos realizados con el objetivo de evaluar la influencia que tiene la codificación utilizada para representar las soluciones en un AG. Las características principales del AG implementado se describen a continuación:

- estrategia de selección $(500 + 500)$,
- operadores de cruce: un punto, dos puntos y uniforme. Probabilidad de cruce $P_x = 0.8$,
- operadores de mutación: para el caso de codificación absoluta se utilizó mutación de un punto; para el caso de relativa, la mutación cambia dos valores consecutivos. Probabilidad de mutación $P_{mut} = 0.3$,

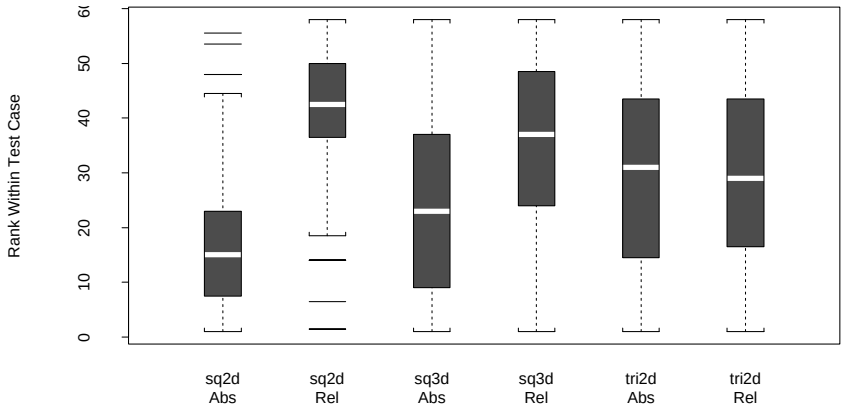


Figura 4.7: *Distribución de los rankings relativos para las codificaciones absoluta y relativa sobre los retículos cuadrado, cúbico y triangular.*

- función de energía: se utilizó la versión modificada presentada en la sección anterior

- instancias de de prueba:

  1. Retículo Cuadrado: $\{s_1, s_7, s_8, s_9, s_{10}\}$ de la Tabla 4.6
  2. Retículo Cúbico: $\{c_7, c_8, c_9, c_{10}, c_{11}\}$ de la Tabla 4.7
  3. Retículo Triangular: $\{t_8, t_9, t_{10}, t_{11}, t_{12}\}$ de la Tabla 4.8

Para distinguir el efecto de la codificación de otros factores como los operadores o el retículo, los experimentos se realizaron sobre tres tipos de retículos: cuadrado, triangular y cúbico. Para cada uno de ellos se eligieron cinco secuencias de diferentes longitudes y dificultad. Sobre cada retículo, cada instancia y cada codificación, se realizaron 29 ejecuciones por cada uno de los tres AG definidos en función del operador de cruce utilizado. De cada ejecución, se registró el valor de energía de la mejor estructura encontrada. Para cada combinación de retículo, operador de cruce e instancia de prueba, se computó el ranking sobre todas las ejecuciones que usaron la codificación relativa y la absoluta. El valor máximo posible es 58, ya que se realizaron 29 experimentos en cada caso.

(a)

	s_1	s_7	s_8	s_9	s_{10}
$1P$	R	R	R	R	R
$2P$	R	R	R	R	R
Ux	R	R	R	R	R

(b)

	c_7	c_8	c_9	c_{10}	c_{11}
$1P$	R	=	=	R	R
$2P$	=	R	R	R	R
Ux	R	R	R	R	R

(c)

	t_8	t_9	t_{10}	t_{11}	t_{12}
$1p$	=	=	A	=	A
$2P$	=	=	=	=	A
Ux	R	R	R	A	A

Tabla 4.3: *Resultados de los t-test: el signo = indica que no existen diferencias significativas en las codificaciones. La R aparece si la codificación relativa resultó mejor que la absoluta, y la A aparece en caso contrario. En (a) se muestran los resultados sobre el retículo cuadrado, en (b) sobre el cúbico y en (c) sobre el triangular.*

En la Figura 4.7 se resumen los resultados del experimento. Se muestran los diagramas de caja de los rankings relativos de los resultados finales para cada retículo y codificación sobre las 5 instancias en conjunto. Cuanto mayor es el ranking, mejor es el método correspondiente. El gráfico muestra claramente que la codificación relativa es, al menos, tan buena como la absoluta para los retículos triangular y cúbico. Para el caso del retículo cuadrado, resulta mucho mejor.

Para evaluar si las diferencias en la media del valor de energía obtenidas para codificaciones son diferentes o no, se realizaron t-tests con un nivel de significación del 95%. Los resultados se presentan en la Tabla 4.3 y de ellos se pueden derivar dos conclusiones: (1) la codificación relativa fue casi siempre mejor que la absoluta para los retículos cuadrado y cúbico (nivel de confianza del 95%); y (2) la robustez de la codificación relativa se degrada en el retículo triangular.

4.4.4 Influencia de la Codificación en *FANS*

En esta sección se presentan los experimentos realizados con el objetivo de evaluar la influencia que tiene sobre *FANS* la codificación utilizada para representar las soluciones [98, 102]. Las pruebas se realizarán sobre tres tipos de retículos: cuadrado, cúbico y triangular, cada uno de ellos con ambas codificaciones: absoluta y relativa. Así, podremos analizar si los resultados con *FANS* coinciden o no con los obtenidos para algoritmos evolutivos.

Implementación de *FANS* para *PSP*

En las secciones siguientes se describen las definiciones utilizadas para cada componente.

Operador de Modificación: para todos los retículos y codificaciones, se utiliza un macro operador $\mathcal{O}$, el cual utiliza un parámetro k que representa el número de posiciones a cambiar en una solución dada. Existen dos modos de operación para el operador $\mathcal{O}$:

- *Modo Segmento*, donde se modifican k posiciones consecutivas de la estructura,
- *Modo Flip*, donde se modifican k posiciones aleatoriamente.

Cuando se utiliza el modo *Segmento*, la porción de estructura seleccionada se puede modificar o reemplazando cada posición por cualquier otra disponible, o realizando una "reflexión" de las posiciones. Por ejemplo, en el retículo cuadrado con codificación absoluta la reflexión provocaría el intercambio de los movimientos $Arriba \leftrightarrow Abajo$ o $Izquierda \leftrightarrow Derecha$. El plano de reflexión se elige aleatoriamente. Para el caso de codificación relativa, el reemplazo se realiza entre los movimientos $TurnRight \leftrightarrow TurnLeft$.

Valoración Difusa "Aceptabilidad": la idea de aceptabilidad es similar a la utilizada en los capítulos anteriores. La definición utilizada es:

$$\mu(q,s) = \begin{cases} 0.0 & \text{if } f(q) < \beta \\ (f(q)-\beta)/(f(s)-\beta) & \text{if } \beta \leq f(q) \leq f(s) \\ 1.0 & \text{if } f(q) > f(s) \end{cases} \tag{4.7}$$

donde f es la función objetivo, s es la solución actual y q es una solución del vecindario operacional. El parámetro β se calcula como $\beta = f(s) * (1-\gamma)$ $\gamma \in [0..1]$. El valor de γ se fijó en 0.2 a partir de observaciones empíricas.

Administrador de Operación: el macro operador $\mathcal{O}$ se adaptará a través de cambios en el parámetro k. Cada llamada al administrador provoca que el valor de k sea decrementado: siendo k_t el valor del parámetro en el instante t, luego $k_{t+1} = k_t - 1$. De esta manera, el operador realizará modificaciones "gruesas" inicialmente (las cuales se corresponden con una etapa de exploración), las cuales se irán haciendo mas ajustadas a medida que la búsqueda progrese. Para estos experimentos el valor inicial de k es $n/4$, siendo n la longitud de la secuencia.

Administrador de Vecindario: para los experimentos usaremos el administrador $First$ presentado anteriormente, el cual retorna la primera solución $x \in \hat{N}(s)$ encontrada utilizando como máximo cierto número de intentos $maxTrials = 50$. Dado que todas las soluciones generadas son evaluadas, el valor del parámetro $maxTrials$ establece un límite superior para el número de evaluaciones de la función de costo disponibles en cada iteración para el administrador de vecindario.

El administrador construye soluciones del vecindario operativo a través de llamadas al operador $\mathcal{O}$ en ambos modos. Cuando la llamada se realiza en modo Segmento, el administrador determina la posición inicial del segmento: el i-esimo llamado a $\mathcal{O}$ modificará el segmento que comienza en la posición $p = (i \ mod \ n) + 1$. Naturalmente se verifica que $p + k < n$. Podemos ver este enfoque como una ventana deslizante de tamaño k moviendose a lo largo de la estructura. Cuando el operador se llama en modo Flip, se modifican k posiciones seleccionadas aleatoriamente y sin restricciones.

Par $(cond, accion)$**:** la condición $HayEstancamiento()$ se hace verdadera cuando el valor del parámetro k del operador se hace cero, o cuando se realizaron $Tope$ iteraciones sin mejoras en la mejor solución encontrada. Esto indica que el algoritmo se esta moviendo entre soluciones "aceptables" pero no se han producido mejoras en las ultimas $Tope$ iteraciones.

En este caso, se ejecuta el procedimiento $Escape()$ el cual genera una nueva solución aleatoria. Esta solución es evaluada, la valoración subjetiva se adapta para reflejar el cambio y el valor del parámetro k del operador se reinicializa en $k = n/4$. Posteriormente, la búsqueda continua desde dicha solución inicial.

Experimentos y Resultados

Para nuestros experimentos se utilizaron 3 valores de $\lambda = \{0.0, 0.9, 1.0\}$ lo cual lleva a 3 comportamientos para *FANS*: λ_0, $\lambda_{.9}$, and λ_1. Cuando *FANS* utiliza $\lambda_{.9}$, algunos movimientos de empeoramiento son aceptados. Por cada valor de λ y cada codificación (relativa y absoluta), se realizaron 30 ejecuciones de cada algoritmo, para un total de $30 * 3 * 2 = 180$ experimentos para cada instancia de prueba. La función de costo utilizada es la versión extendida presentada en [76] y cada experimento utiliza como máximo 10^5 evaluaciones de la misma. Las instancias de prueba son las siguientes:

- Retículo Cuadrado: $\{s_1, s_2, \ldots, s_7\}$ de la Tabla 4.6
- Retículo Cúbico: $\{c_1, c_2, \ldots, c_6\}$ de la Tabla 4.7
- Retículo Triangular: $\{t_1, t_2, \ldots, t_7\}$ de la Tabla 4.8

Para detectar si las diferencias en los valores medios obtenidos por los distintos algoritmos eran significativas o no, se realizaron t-test. En la Tabla 4.4 se reporta para cada instancia s_i, t_i, c_i (del retículo cuadrado, triangular y cúbico, respectivamente) y valor de λ, la codificación que permitió obtener los mejores valores en media de la función objetivo. El signo '=' indica que no se detectaron diferencias significativas, la 'R' indica que la codificación relativa fue mejor ($p < 0.05$), mientras que una 'A' aparece si la codificación absoluta resultó la mejor.

(a)

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
λ_0	R	R	R	R	R	R	R
$\lambda_{.9}$	R	R	R	R	R	R	R
λ_1	R	R	R	R	R	R	R

(b)

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
λ_0	R	R	R	R	R	R	R
$\lambda_{.9}$	R	R	=	A	A	=	=
λ_1	R	R	=	A	A	=	A

(c)

	c_1	c_2	c_3	c_4	c_5	c_6
λ_0	R	R	R	R	R	R
$\lambda_{.9}$	=	=	=	=	=	=
λ_1	A	=	=	=	R	=

Tabla 4.4: *Resultados de los t-test: el signo = indica que no existen diferencias significativas en las codificaciones. La R aparece si la codificación relativa resultó mejor que la absoluta, y la A aparece en caso contrario. En (a) se muestran los resultados sobre el retículo cuadrado, en (b) sobre el triangular y en (c) sobre el cúbico.*

Los resultados muestran que cuando se usa una búsqueda tipo caminos aleatorios, λ_0, la codificación relativa permite obtener mejores resultados que la absoluta en los tres retículos. Cuando se utiliza el esquema $\lambda_{.9}$, que permite transiciones a soluciones peores, la codificación relativa resulta superior en el retículo cuadrado, mientras que ambas codificaciones resultan similares en los otros dos. Finalmente, cuando *FANS* se comporta como un multi-start hill-climber, λ_1, la codificación absoluta resulta superior sobre el retículo triangular, la relativa es mejor sobre el cuadrado y ambas codificaciones resultan similares sobre el retículo cúbico.

4.4.5 Evitando los problemas del *AG* en *PSP*

En secciones anteriores se describió que la aplicación de *AG*'s con operadores de cruce estandar para *PSP* presentaban un problema "teórico". La principal conclusión de dichos trabajos fué que los mismos resultados que obtenía el

I	Algor.	Media	DT	Max.
s_1	*FANS*	8.78	0.43	9.01
	AG	7.60	0.93	9.01
s_2	*FANS*	6.61	0.62	8.01
	AG	6.83	0.65	8.01
s_3	*FANS*	16.48	1.08	19.02
	AG	16.74	1.29	20.01
s_4	*FANS*	8.24	0.43	9.01
	AG	7.91	0.48	9.01
s_5	*FANS*	7.71	0.47	8.01
	AG	7.14	0.35	8.01
s_6	*FANS*	3.23	0.43	4.00
	AG	2.90	0.48	4.00
s_7	*FANS*	9.68	0.55	10.01
	AG	8.17	1.02	10.01

Tabla 4.5: *Medias, desviaciones y mejores valores obtenidos por cada algoritmo en cada instancia (sobre 30 ejecuciones).*

AG podían alcanzarse potencialmente eliminando la población y mediante la aplicación de mutaciones sobre un único individuo. En esta sección se describen los experimentos realizados para mostrar que *FANS* es, al menos, tan bueno como un *AG* en la resolución de *PSP* dando soporte experimental a la hipótesis anterior.

En primer término se presentan las características principales del algoritmo genético utilizado. La representación para los individuos es la misma que la utilizada en *FANS*. La población inicial esta formada por soluciones aleatorias representando conformaciones sin cruces (self avoiding). El operador de mutación es igual al operador de modificación utilizado en *FANS*. Dado que no existe el administrador de operación en el *AG*, el valor del parámetro k (número de posiciones a modificar) se selecciona aleatoriamente en el intervalo $[1, n/4]$. La posición inicial para el segmento, también se elige aleatoriamente

en el intervalo $[1, n-k]$. La mutación se aplica a todos los hijos con probabilidad $P_m = 0.25$. Se utilizó cruce de 2 puntos con probabilidad $P_x = 0.8$. Finalmente, para la selección, se utilizó selección por torneos con tamaño $q = 2$ en un esquema $(\mu = 200, \lambda = 350)$. También se utiliza elitismo.

Los experimentos se realizaron sobre el retículo cuadrado, con codificación relativa, sobre las instancias de prueba $s_1, \ldots, s_7$ mostradas en la Tabla 4.6. Teniendo en mente los resultados previos, se utilizó la misma implementación de *FANS* con $\lambda = 0.9$. Nuevamente, para cada instancia y algoritmo, se realizaron 30 ejecuciones, cada una finalizando cuando se utilizaron 10^5 evaluaciones de la función de costo. Al algoritmo genético se le permite finalizar la iteración correspondiente.

Los resultados aparecen en la Tabla 4.5, donde se reportan la media, desviación estandar y mejor valor obtenido sobre las 30 ejecuciones para cada instancia y algoritmo. En términos de las medias, *FANS* obtiene mejores resultados en 5 casos sobre 7. En general, la desviación estandar es menor en *FANS* que en AG y ambos algoritmos alcanzaron el óptimo correspondiente de todas las instancias, salvo $I5$ donde el AG alcanzó el mejor valor.

4.4.6 Instancias de prueba utilizadas en los experimentos

Nro	Secuencia	Long	Opt
s_1	$HPHPPHHPHPPHPHHPPHPH$	20	-9
s_2	$PPHPPHHPPPPHHPPPPHHPPPPHH$	25	-8
s_3	$P^2H(P^2H^2)^2P^5H^{10}P^6(H^2P^2)^2HP^2H^5$	48	-22
s_4	$PHPPHPHHHPHHPHHHHH$	18	-9
s_5	$HPHPHHHPPPHHHHPPHH$	18	-8
s_6	$HHPPPPPHHPPPHPPPHP$	18	-4
s_7	$HHHPPHPHPHPPHPHPHPPH$	20	-10
s_8	$PPPHHPPHHPPPPPHHHHHHHHPPHHPPPPHHPPHPP$	36	-14
s_9	$H^2(PH)^4H^3P(HP^3)^2HP^4(HP^3)^2HPH^4(PH)^4H$	50	-21
s_{10}	$H^12(PH)^2(P^2H^2)^2(PPH)^2(HP^2H)^2(P^2H^2)^2P^2(HP)^2H^{12}$	64	-42

Tabla 4.6: *Instancias de prueba para el retículo cuadrado*

Nro	Secuencia	Long	Opt
c_1	$PPHPPHHPPPPHHPPPPHHPPPPHH$	25	-8
c_2	$PPPHHPPHHPPPPPHHHHHHHHPPHHPPPPHHPPHPP$	36	-14
c_3	$HHHPHPHPHPHPHPHPHPHPHHH$	23	-25
c_4	$HHPPHPPHPPHPPHPPHPPHPPHH$	24	-17
c_5	$HHHPPHPPHPPHPPHPHPPHPHPPHPPHHH$	30	-25
c_6	$HHHPPHPPHPPHPHPPHPHPPHPPHPPHHH$	30	-25
c_7	$HPH^2P^2H^4PH^3P^2H^2P^2HPH^3(PH)^2HP^2H^2P^3HP^8H^2$	48	-32
c_8	$(PH)^2P^4(HP)^3(PH)^2H^5P^2H^3(PHP)^2H^2P^2HPH^3P^4H$	48	-34
c_9	$P^2H^5P(P^2H^2P^3)^2HP^5(PH)^2P(P^2H)^3P^5HP^4(H^2P)^2(PHP)^2$	64	.
c_{10}	$HPH^2P^2H^2PHP^5H^3PH^4(P^2H)^2PH^2P^3(HP)^2PH^3PH^2PHP^5H^8P^3$	64	.
c_{11}	$(P^2H)^3H^2P^3HP(HP^2)^3P^4HP^2H^3(P^2H)^3PHP^6H^3P^5(HP)^2$	64	.

Tabla 4.7: *Instancias de prueba utilizadas para el retículo cúbico*

Nro	Secuencia	Long	Opt
t_1	$HHPHPHPHPHPH$	12	-11
t_2	$HHPHPPHPPHPPHPPH$	16	-11
t_3	$HHPPHPPHPPHPPHPPH$	17	-11
t_4	$HHPPHPPHPHPPHPHPPHPHH$	21	-17
t_5	$HHPPHPPHPHPHPPHPPHPPHH$	22	-17
t_6	$HHHPHPHPPHPHPHPHPHPHPHHH$	24	-25
t_7	$HHHPPHPPHPPHPPHPHPPHPHPPHPPHHH$	30	-25
t_8	$HHPHPHPHPHPHPHPHH$	17	-17
t_9	$HHPHPPHPPHPHPHPPHPPHH$	21	-17
t_{10}	$HHHPHPHPHHPHPHPHPHPHPHHH$	23	-25
t_{11}	$HHHPPHPPHPPHPPHPHPPHPHPPHPPHHH$	30	-25
t_{12}	$H^4P^3(HP)^2(PH)^8P^2H^{16}P^3(HP)^4(PH)^2H^9P^2(HP)^4P(PH)^2H^9P^3H^6$	100	.

Tabla 4.8: *Instancias de prueba utilizadas para el retículo triangular*

4.5 *FANS* para el Problema de Emparejamiento Estructural de Moléculas

En esta sección abordaremos el problema de emparejamiento estructural de moléculas (*MSM*) utilizando el modelo presentado en [10, 11, 12, 70] y cuyo objetivo es minimizar la *disimilaridad* entre las posiciones de los atomos de dos moleculas A y B.
Siendo Na, Nb los tamaños de dichas moléculas, en la práctica se pueden reconocer 4 casos relevantes:

- $Na = Nb$, donde la molécula A completa, se compara con toda la molécula B,
- $Na \leq Nb$, donde se busca en B una región que se parezca a una zona definida de A,
- $Na < Nb$, donde se busca en B una región que se parezca a toda la molécula A,
- $Na \leq Nb$, este es el caso más general, y se busca en B una región que se parezca a alguna zona sin especificar de A

En esta memoria nos centraremos en el caso 3, donde podemos considerar a la molécula A como un patrón y deseamos encontrar en B un subconjunto de átomos que sea lo más "parecido" a A. Se plantea como objetivo, analizar el comportamiento de *FANS* sobre el problema *MSM* en dos aspectos:

1. analizar la influencia del tamaño del patrón A, y
2. analizar la influencia del parámetro λ utilizado para controlar la selección de nuevas soluciones.

Para ello, se realizará una serie de experimentos sobre datos artificiales con óptimo conocido para conseguir una idea clara acerca de la viabilidad de *FANS* como herramienta para la resolución de *MSM*.

4.5.1 Definición del Problema

El problema de *MSM* que se intenta resolver se define de la siguiente manera: dadas dos moléculas A, B y las correspondientes matrices de distancias intra atomos D_A, D_B, donde $D_C(i, j)$ es la distancia euclídea entre los atomos i y j en la molécula C, el objetivo es encontrar una matriz de permutación P de filas/columnas que minimice

$$Min.\ ||D_A - P^T D_B P|| \tag{4.8}$$

En otras palabras, estamos buscando un subconjunto de puntos de B cuyo conjunto de intradistancias sea el más similar respecto a A. La norma solo se calcula para los átomos seleccionados y resulta claro que es un problema combinatorio complejo ya que, en primer lugar hay que determinar un subconjunto de átomos de B, y posteriormente se requiere establecer cierto orden para ellos.

La comparación de proteínas mediante matrices de distancia se realizó previamente en [59]. Una matriz de distancias es una representacion en dos dimensiones de una estructura tridimensional. Es independiente del espacio de coordenadas y contiene suficiente información para reconstruir la estructura tridimensional mediante técnicas de geometría de distancias.

4.5.2 Implementación de *FANS* para *MSM*

En las secciones siguientes se describen las definiciones utilizadas para los componentes de *FANS*.

Representación de Soluciones: una solución representa una matriz de permutación; es decir, una matriz cuadrada que verifica

$$\sum_{i=1}^{n} p_{ij} = 1 \text{ para todo } j = 1, 2, ..., n$$
$$\sum_{i=1}^{n} p_{ji} = 1 \text{ para todo } j = 1, 2, ..., n$$
$$p_{ji} = 0, 1 \text{ para todo } i, j = 1, 2, ..., n$$

Cada fila y columna tienen un único valor en 1. Este hecho se aprovecha para representar la matriz en espacio $\mathcal{O}(n)$ siendo n el tamaño de la molécula mayor. Por ejemplo, la permutación

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

se representa como un vector $P = \{3, 6, 4, 1, 2, 5\}$ donde la componente $P[i]$ indica la posición del valor 1 en la fila i. Extendiendo el ejemplo, supongamos un patrón $A = \{a_1, a_2, a_3\}$ de tamaño $p = 3$ y una proteína $B = \{b_1, b_2, b_3, b_4, b_5, b_6\}$ de tamaño $n = 6$. La permutación anterior "reordena" B de la siguiente manera: $B = \{b_3, b_6, b_4, b_1, b_2, b_5\}$.

La función de costo solo se aplica a los primeros p atomos, los cuales forman el llamada *conjunto interno*. Los restantes $n - p$ atomos forman el *conjunto externo*. En esta solución, los atomos emparejados son $M = \{(a_1, b_3), (a_2, b_6), (a_3, b_4)\}$.

Operador de Modificación: el operador $\mathcal{O}$ construye nuevas soluciones (permutaciones) a partir de cambios sobre cierta solución s de referencia. Dados un parámetro k y una solución s, el operador genera nuevas soluciones mediante la aplicación de alguna de las siguientes operaciones:

- 2-*Swap*: se intercambian en la permutación, los valores de las posiciones (i, j) (seleccionadas al azar). Este procedimiento se repite k veces y no

se permite el intercambio de dos posiciones que pertenezcan al conjunto externo ya que la solución no sería modificada.

- *MoveBlock*: se seleccionan e intercambian dos bloques de longitud k seleccionados al azar. Por ejemplo, si $k = 2$ y $s = \{1, 2, 3, 4, 5\}$, entonces una posible solución es $\hat{s} = \{4, 5, 3, 1, 2\}$. El primer bloque siempre pertenece al conjunto interno.

- *InvertBlock*: se invierten los valores en un bloque de longitud k seleccionado al azar. Por ejemplo, si $k = 2$ y $s = \{1, 2, 3, 4, 5\}$, entonces una posible solución es $\hat{s} = \{3, 2, 1, 4, 5\}$. La posición inicial del bloque pertenece al conjunto interno.

Las operaciones se seleccionan siguiendo probabilidades predefinidas. La operación 2-*Swap* se selecciona con una probabilidad de 0.7. Si no resulta seleccionada, entonces las operaciones $MoveBlock$ o $InvertBlock$ tienen las mismas probabilidades de ser seleccionadas.

El valor del parámetro k se modifica en función del estado de la búsqueda. La estrategia de modificación es responsabilidad del administrador de operación y se describe a continuación.

Administrador de Operación: la estrategia de variación del parámetro k es muy simple: cada vez que el administrador se ejecute, se decrementará el valor de k en una unidad. De esta manera, el algoritmo comenzará realizando modificaciones grandes sobre la solución, lo que se corresponde con una etapa de exploración. A medida que la ejecución avance, el valor de k se reducirá lo cual permitirá realizar un "ajuste fino" de la solución; es decir, explotación.

Valoración Difusa: para este problema, las soluciones se evalúan en términos de la noción de "Acceptabilidad" con una semántica similar a la utilizada en otros capítulos de esta memoria. A efectos de completitud, se presenta a continuación la función de pertenencia asociada:

$$\mu(q,s) = \begin{cases} 1.0 & \text{if } f(q) < f(s) \\ \frac{(f(q)-\beta)}{(f(s)-\beta)} & \text{if } f(s) \leq f(q) \leq \beta \\ 0.0 & \text{if } f(q) > \beta \end{cases}$$

donde f es la función objetivo, s es la solución actual, q es una solución generada por el operador, y β representa el límite para lo que se considera aceptable. El valor de β se define como $\beta = f(s) * (1 + \alpha)$, y en este caso se utilizó $\alpha = 0.01$.

Administrador de Vecindario: por simplicidad, se utiliza el administrador de vecindario $First$ el cual construye soluciones mediante el operador $\mathcal{O}$, hasta encontrar una solución aceptable. La búsqueda continúa mientras no se hayan superado $maxTrials = 600$ intentos, en cuyo caso se genera una condición de excepción.

Cuando el administrador de vecindario recibe un operador con $k = 1$, entonces se ejecuta un macro operador denominado $BuscarUno$ el cual busca una solución aceptable de forma sistemática: se selecciona aleatoriamente una posición i y luego se evalúa cada posible intercambio $(i, j), j > i$. El proceso se repite si es necesario para todo i. En el peor caso, este operador tiene una complejidad de $\mathcal{O}(n^2)$.

4.5.3 Experimentos y Resultados

Para resolver el problema *MSM* con *FANS*, se utilizarán proteínas como ejemplo de moléculas. Como atomos solamente se considerarán los carbonos centrales C_α de cada residuo de aminoácido.

El problema de comparación de estructuras no tiene una única respuesta y, más aún, la definición de una medida de similaridad entre proteínas es un tema sobre el que no existe acuerdo general (mirar por ejemplo [72] y las referencias allí citadas) Es por eso que la evaluación de *FANS* se realiza sobre problemas de prueba artificiales pero con solución óptima conocida.

Para construir el conjunto de pruebas se procede de la siguiente manera: dada una proteína B con n atomos, se selecciona aleatoriamente un subconjunto de ellos de tamaño p. Este subconjunto representa el patrón A. Luego

Proteína B	n
101M	154
1ALB	131
1FMB	104
2LZM	164
3DFR	162
3HHB	141

Tabla 4.9: *Proteínas para representar B*

el objetivo consiste en encontrar el patrón A en B, lo cual implica "elegir" el subconjunto correcto de atomos de B y determinar el orden correcto para los mismos: siendo n el tamaño de la proteína y p el del patrón, existen $\binom{n}{p} * p!$ formas de elegir p atomos de n, cada uno con las $p!$ permutaciones posibles. Las instancias construídas de esta forma, tienen un valor óptimo de cero.

Para los experimentos se seleccionaron 6 proteínas del *PDB*[8]; sus códigos y tamaños (en cantidad de carbonos centrales) se muestran en la Tabla 4.9. Estas proteínas representan a B. Luego, para cada proteína se definieron 4 tamaños de patrón correspondientes al $pSize = \{20, 40, 60, 80\}\%$ del tamaño de B. Para cada tamaño $pSize$, se generaron 15 patrones, es decir 15 subconjuntos de puntos seleccionados aleatoriamente. Finalmente, para cada valor de patrón, proteína y valor de $\lambda = \{0.7, 0.8, 0.9, 1.0\}$ se realizaron hasta 3 ejecuciones de *FANS* para intentar encontrar el patrón (obtener el óptimo). Cada ejecución finaliza cuando se realizaron cierto número de evaluaciones de la función de costo, o en forma equivalente, cuando se analizaron cierto número de configuraciones. Este valor se calcula en función de los tamaños de la proteína y patrón utilizados y se define como $maxEvals = (n + \frac{p}{2}) * 1200$.

Al final de cada ejecución, se registraron tres valores:

- $Best$: el costo de la mejor solución encontrada,

[8]Protein Data Bank es una base de datos pública (y gratuita) de estructuras de proteínas. Se accede a través de http://www.rcsb.org

	λ			
Prot	.70	.80	.90	1.00
101M	42.79	48.58	53.43	54.74
1ALB	36.51	48.16	40.46	46.66
1FMB	80.02	71.01	72.62	75.39
2LZM	64.86	65.96	62.36	68.08
3DFR	80.82	93.19	89.47	92.32
3HHB	52.09	60.15	51.67	56.97

Tabla 4.10: *Media del Porcentaje de atomos emparejados correctamente en función de* λ.

- $pctOK$: el porcentaje de atomos emparejados correctamente, lo cual es una medida de eficiencia,
- $e2best$: el porcentaje de evaluaciones de la función de costo utilizado para encontrar la mejor solución, lo cual es una medida del esfuerzo.

El análisis de resultados se realizó para cada proteína del conjunto de prueba y en forma global sobre todo el conjunto.

Resultados por Proteína

En primer lugar se analizó el valor promedio de $pctOK$ por proteína y valor del parámetro λ. Estos resultados aparecen en la Tabla 4.10. El primer aspecto a destacar es que no existe un único valor de λ que resulte mejor para todos los casos. Por ejemplo, para la proteína 101M el mejor valor se obtuvo con $\lambda = 1.0$, mientras que para 1FMB fue $\lambda = 0.7$ y para 3HHB fue $\lambda = 1.0$. En cierta forma esto es esperable, ya que cada instancia de prueba presenta sus propias características; es decir la distribución espacial de los atomos puede provocar la aparición de simetrías internas o conglomerados particulares de puntos que en algunos casos puede engañar o ayudar al algoritmo para obtener soluciones de alta calidad.

Los resultados en la Tabla 4.11 muestran la media de $pctOK$ para cada proteína y cada tamaño de patrón. Se observa que, en general, existe una

	$pSize$			
Prot	.20	.40	.60	.80
101M	49.18	51.08	48.55	50.31
1ALB	37.74	40.93	44.43	47.96
1FMB	69.03	69.26	78.11	82.96
2LZM	65.34	60.40	64.61	71.29
3DFR	79.61	90.22	93.47	92.21
3HHB	52.37	54.73	53.75	59.87

Tabla 4.11: *Media del Porcentaje de atomos emparejados correctamente en función del tamaño del patrón pSize*

relación directa entre el valor promedio de $pctOK$ y el tamaño del patrón. Los patrones más grandes parecen resultar más simples.

Para confirmar la calidad de los resultados obtenidos por *FANS* sobre este problema, se analizó a continuación la cantidad de patrones que no fueron resueltos. Se recuerda que existen 15 patrones para cada tamaño y se considera que un patrón fue resuelto si al menos uno de los tres intentos realizados finalizó con $pctOK \geq 95$.

En la Tabla 4.12 se muestra el número de patrones (de un total de 15) que NO fueron resueltos ($pctOK < 95$) en al menos tres intentos. Los resultados se encuentran discriminados por tamaño del patrón (en la primera columna) y valor de λ (en la primera fila). El valor total en las filas y columnas es 60.

Las tablas permiten establecer que *FANS* resulta altamente efectivo. Dejando a un lado los resultados para la proteína 1ALB, casi todos los patrones pudieron ser resueltos. Por ejemplo, para el caso 101M, un tamaño de patrón del 40%, y un valor de $\lambda = 0.7$, *FANS* no pudo resolver 5 patrones. Sin embargo, cuando se utilizó $\lambda = 0.9$, solamente quedó 1 patrón sin resolver.

Esta situación se repite aproximadamente para el resto de las proteínas, donde para cada tamaño de patrón, existe un valor para λ que permite resolver mas de 12 patrones. Es decir, solo quedan sin resolver 3 o menos patrones.

Respecto a la dificultad de cada proteína del conjunto de prueba, el caso más simple lo representa 3DFR. Para cualquier valor de λ utilizado, *FANS* fue

capaz de resolver todos los patrones. En el otro extremo se encuentra 1ALB, donde cada valor de λ permite obtener resultados muy diversos. Por ejemplo, para un tamaño de patrón del 20%, el rango de patrones no resueltos varía entre 3 (con $\lambda = 0.8$) y 8 (con $\lambda = 0.9$). En general, se puede observar que para cada tamaño de patrón quedan sin resolver aproximadamente 20 sobre 60 casos (un 33%).

Análisis Global de Resultados

Con el objetivo de obtener conclusiones más generales, se analizaron también los resultados sobre todo el conjunto de prueba.

En las Figuras 4.8 y 4.9 se muestran histogramas en los cuales el eje x indica el valor de $pctOK$ mientras que el eje y indica el número de ejecuciones de *FANS* que finalizaron en dicho valor. La Figura 4.9 muestra la distribución de casos agrupados por tamaño del patrón y la Figura 4.8 agrupador por valor de λ. El ancho de cada columna es de 2.5%.

Se pueden destacar dos elementos que resultan evidentes. Primero, no parece existir relación entre los valores de λ y $pctOK$; la distribución de los casos en los 4 histogramas es practicamente idéntica. En segundo lugar, existe una relación directa entre el tamaño del patron y valor de $pctOK$, al menos en la cantidad de ejecuciones que finalizaron con más del 95% de atomos correctamente emparejados. Los patrones grandes se resuelven más facilmente que los pequeños.

Esto parece una contradicción si se analiza el número de potenciales soluciones para cada tamaño de patrón. Si consideramos una proteína B con tamaño 100 y dos patrones P_1, P_2 con tamaños 20 y 80, luego el número de combinaciones para cada par es $(B, P_1) = \binom{100}{20} * 20!$ y $(B, P_2) = \binom{100}{80} * 80!$ respectivamente. Es decir $(B, P_1) < (B, P_2)$ y el patrón P_2 resulta más simple de resolver que el P_1.

Esta situación se explica partiendo del hecho que existen pocos patrones "grandes" posibles en una proteína que difieran mucho entre sí. Supongamos una proteína con 100 atomos y dos patrones P_1, P_2 de tamaño 80. El patrón P_2 contendrá como máximo 20 atomos diferentes (los que no estaban en P_1).

101M

%/λ	0.7	0.8	0.9	1	**T**
20	1	5	3	2	11
40	5	2	1	2	10
60	4	4	2	2	12
80	7	2	4	2	15
T	17	13	10	8	

1ALB

%/λ	0.7	0.8	0.9	1	**T**
20	7	3	8	4	22
40	6	5	6	5	22
60	8	4	4	5	21
80	7	3	6	2	18
T	28	15	24	16	

1FMB

%/λ	0.7	0.8	0.9	1	
20	2	1	1		4
40	1		2	2	5
60					
80		1	1		2
T	3	2	4	2	

2LZM

%/λ	0.7	0.8	0.9	1	**T**
20		2	3	4	9
40	3	2	3	2	10
60	4	2	1	3	10
80	2	1	1	1	5
T	9	7	8	10	

3DFR

%/λ	0.7	0.8	0.9	1	
20	1				1
40					
60					
80					
T	1				

3HHB

%/λ	0.7	0.8	0.9	1	**T**
20	4	2	4	3	13
40	1	1	5	4	11
60	3	4	4	2	13
80	4	1	1	1	11
T	12	8	14	10	

Tabla 4.12: *Número de patrones no resueltos (sobre un total de 15) por cada valor de λ y tamaño del patrón.*

Por lo tanto, cuando *FANS* tiene que resolver patrones grandes, el problema de seleccionar el subconjunto correcto se reduce y basicamente se transforma en el problema de encontrar la permutación correcta. Dado que el número de permutaciones posibles aumenta con el tamaño del patrón, y *FANS* resuelve mejor patrones grandes que pequeños, se pueden establecer dos cosas: la primera es la habilidad de *FANS* para resolver problemas en espacios de búsqueda de gran tamaño y la segunda, que determinar el subconjunto correcto es más complejo que determinar la permutación correcta.

Los histogramas (Figs. 4.9 y 4.8) revelan además una situación interesante, ya que se puede observar una "brecha" en los valores de $pctOK \in [0.25...0.75]$. Practicamente ninguna ejecución del algoritmo, finalizó con tales valores de atomos emparejados correctamente. Esto implica que el algoritmo alcanza soluciones con un valor alto de $pctOK$, o se "desvía" hacia soluciones que contienen un alto número de emparejamientos incorrectos. En principio, esta situación la atribuímos a simetrías internas de las estructuras que pueden guiar al algoritmo hacia optimos locales de buena calidad, pero que no representan el óptimo buscado. Una situación similar se reporta en [11] donde los autores atacaban un problema similar al aquí presentado mediante recocido simulado.

La forma más simple de evitar esta especie de dicotomía, consiste en realizar varias ejecuciones de *FANS*, cada una partiendo desde soluciones iniciales diferentes. Este fue el enfoque aplicado en este trabajo y en vista de los resultados podemos considerarlo útil: practicamente ningún patrón quedó sin resolver utilizando un máximo de tres ejecuciones del algoritmo.

En todo el experimento se realizaron un total de 3022 ejecuciones, de las cuales 1514 (50%) pueden considerarse satisfactorias (es decir, finalizaron con $pctOK > 95\%$). Con el óptimo, $pctOK = 100\%$, finalizaron 998 (33%) ejecuciones. En la Tabla 4.13 se muestra la distribución de estas 1514 ejecuciones en términos de λ y tamaño de patrón $pSize$. Naturalmente, y en consonancia con los resultados previos, los valores más altos corresponden a los patrones de mayor tamaño. Respecto a los valores en función de λ, se detecta una ligera ventaja para el caso $\lambda = 1.0$.

	λ				
psize	.70	.80	.90	1.00	Total
.20	89	89	79	88	345
.40	92	94	86	95	367
.60	90	91	103	94	378
.80	95	109	97	123	424
Total	366	383	365	400	1514

Tabla 4.13: *Distribución de casos exitosos por valor λ y tamaño del patrón pSize.*

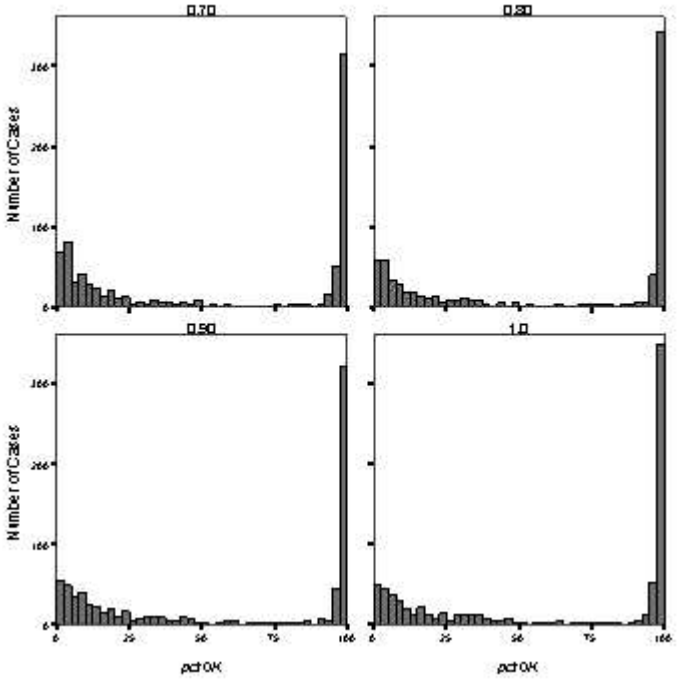


Figura 4.8: *Histogramas de los valores de pctOK discriminados por valor de λ.*

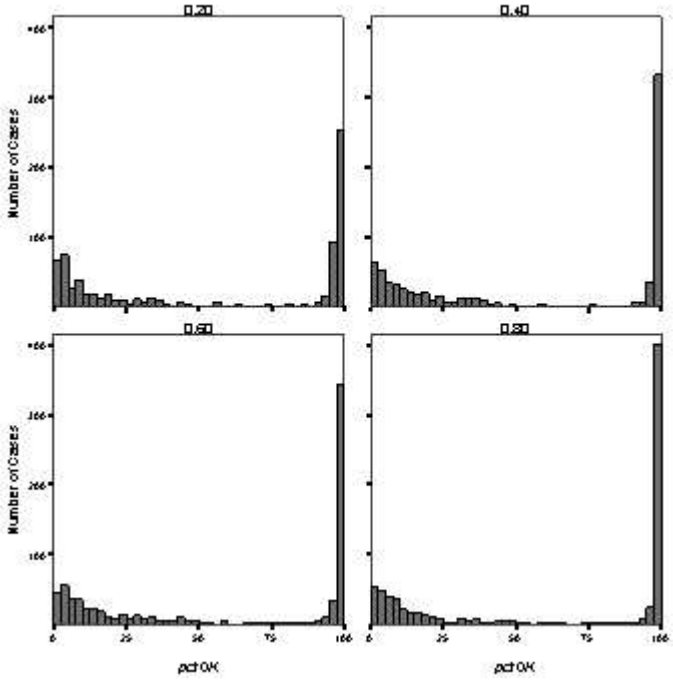


Figura 4.9: *Histogramas de los valores de pctOK discriminados por tamaño de patrón.*

$pSize/\lambda$	0.7	0.8	0.9	1	**Total**
0.2	54.24 (49.53)	56.37 (50.18)	56.25 (48.57)	58.34 (51.43)	56.26 (49.96)
0.4	61.75 (49.50)	63.30 (53.64)	59.37 (45.41)	61.99 (51.22)	61.58 (50.05)
0.6	67.68 (54.15)	67.99 (59.81)	63.94 (51.18)	66.31 (52.77)	66.52 (54.36)
0.8	72.33 (59.47)	66.21 (57.12)	64.31 (55.18)	66.43 (60.08)	67.42 (58.06)
Total	64.05 (53.24)	63.53 (55.29)	60.91 (50.32)	63.29 (54.35)	62.96 (53.35)

Tabla 4.14: *Porcentaje promedio del número de evaluaciones necesarias para alcanzar la mejor solución. Los valores entre paréntesis indican el mismo valor calculado sobre el subconjunto de ejecuciones exitosas ($pctOK > 95\%$).*

Análisis en términos de la variable $e2Best$

El último elemento a analizar, es el esfuerzo utilizado para obtener la mejor solución en cada ejecución. Este esfuerzo fue medido a través del número de evaluaciones de la función de costo utilizadas para alcanzar dicha mejor solución. Dado que el límite de evaluaciones disponibles para el algoritmo se calculó en función de la instancia de prueba, se utilizó el porcentaje de evaluaciones utilizadas sobre el total disponible.

En la Tabla 4.14 se muestran dichos porcentajes sobre el conjunto de todas las ejecuciones. Entre paréntesis, aparecen los resultados obtenidos sobre el subconjunto de las 1514 ejecuciones exitosas ($pctOK > 95\%$). Respecto a la relación entre esfuerzo y tamaño del patrón, resulta evidente que a mayor tamaño de patrón, se utilizan mas evaluaciones. En términos del parámetro λ, no se detecta una relación clara con el esfuerzo.

Analizando solamente los valores para los casos exitosos, se puede observar que (en promedio) en ningún caso se utilizaron más del 60% de las evaluaciones disponibles. Esto implica que si ese límite para el esfuerzo se sobrepasa y el óptimo no fue encontrado, entonces sería factible aplicar algun criterio de parada heurístico o un mecanismo de reinicialización. Naturalmente, en problemas reales este enfoque no es directo, pero creemos que es posible aplicar alguna clase de aprendizaje que, a partir de información recolectada en sucesivas ejecuciones del algoritmo, permita detectar cuando una ejecución es prometedora o no, de forma similar al esquema planteado en [108].

4.6 Conclusiones

En este capítulo abordamos la aplicación de *FANS* sobre problemas de Bioinformática. Por razones de completitud, en primer lugar introducimos algunos conceptos esenciales como gen, cromosoma, genoma, etc y describimos brevemente el Proyecto Genoma Humano. Luego llegamos a establecer qué es, en nuestra opinión, y qué problemas abarca la Bioinformática. Posteriormente definimos las características principales de los dos problemas de ejemplo que abordamos y se citaron otros trabajos relevantes.

En la sección 4.4, utilizamos *FANS* para analizar dos aspectos importantes del problema de predicción de estructura en el modelo HP. Primero analizamos algunos elementos relevantes de cara al diseño de algoritmos para *PSP* y luego evaluamos la influencia que tiene la utilización de codificación absoluta y relativa sobre el comportamiento de un AG y de *FANS*. Los resultados obtenidos en los experimentos realizados están en consonancia con los obtenidos previamente por [76] y nos permiten concluir que: los algoritmos que utilizan codificación relativa sobre retículo cuadrado, obtienen mejores resultados que aquellos que utilizan la absoluta; las diferencias se reducen sobre el retículo cúbico y se detecta una ligera diferencia a favor de la codificación absoluta en el retículo triangular.

Estos experimentos también nos permitieron detectar cuál es el mejor esquema para futuras aplicaciones de *FANS* sobre *PSP*: un esquema que utiliza codificación relativa, en conjunto con un régimen de búsqueda donde se permitan transiciones a soluciones peores ($\lambda_{.9}$). Es interesante notar que este hecho también fue resaltado para algoritmos meméticos por [81], donde se vió que las etapas de búsqueda local resultaban más productivas si permitían "empeorar" las soluciones.

Creemos que la verificación de ambos aspectos es un paso importante para la comprensión de la aplicabilidad de meta heurísticas para este problema.

En segundo lugar, se había mostrado que la aplicación de algoritmos genéticos con operadores de cruce estandar para *PSP* presentaban un problema desde un punto de vista "teórico". La principal conclusión de dichos trabajos era que los mismos resultados que obtenía un AG con esas carac-

terísticas podían alcanzarse, potencialmente, eliminando la población y realizando mutaciones sobre un único individuo. La comparación de *FANS* frente a un AG en términos de optimización resultó altamente satisfactoria: en este trabajo pudimos obtener los mismos resultados que un AG utilizando un esquema relativamente simple, verificando por lo tanto, la hipótesis que la población puede ser descartada si el AG utiliza un operador de cruce estandar y una representación basada en coordenadas internas.

En nuestra opinión, las conclusiones obtenidas en los dos aspectos analizados resultan relevantes de cara a la comprensión y el diseño de mejores heurísticas para este problema.

Posteriormente, en la sección 4.5 abordamos la resolución del problema de emparejamiento estructural de moléculas mediante *FANS*. Utilizando un conjunto de pruebas artificial, se analizaron dos factores importantes que podían afectar el rendimiento de *FANS* sobre este problema: la influencia del tamaño del patrón a buscar, y la influencia del valor λ utilizado para determinar la aceptación de nuevas soluciones.

El análisis del primer factor reveló que, en promedio, el algoritmo fue capaz de obtener mejores soluciones para patrones grandes que para pequeños. Teniendo en cuenta el número de posibles soluciones en cada caso, los resultados pueden considerarse como una prueba de que determinar el subconjunto correcto de átomos es más complejo que determinar la permutación correcta de los mismos; y que *FANS* es capaz de resolver problemas en espacios de búsqueda de gran tamaño. Como era esperado, a medida que el tamaño del patrón se incrementó, el esfuerzo necesario para resolverlo también aumentó.

Sobre la influencia del parámetro λ, no se pudo obtener evidencia suficiente para sugerir algún valor específico para futuras aplicaciones de *FANS*. Sin embargo, en términos de optimización, consideramos los resultados como muy prometedores: para cada patrón, siempre existió un valor de λ (versión de *FANS*) que permitió obtener una solución exitosa en, a lo sumo, tres ejecuciones del algoritmo. Es decir, podemos concluir que el conjunto de comportamientos que presenta *FANS* con las definiciones provistas, fué suficiente para resolver satisfactoriamente las instancias del conjunto de prueba utilizado.

Conclusiones y Trabajo Futuro

En este apartado se reseña brevemente el trabajo realizado y se detallan las principales aportaciones de esta memoria. También se indican algunas líneas de trabajo futuras, algunas de las cuales ya se encuentran en desarrollo.

Resumen del Trabajo y Conclusiones

El primer capítulo de la memoria estuvo destinado a introducir los elementos esenciales de la disciplina de la optimización combinatoria y las técnicas heurísticas. Se describieron propiedades de las metaheurísticas y se presentaron algunos ejemplos de las mismas. Dada la amplitud de información sobre el tema y dado que su relevamiento exhaustivo estaba fuera de los objetivos de este trabajo, se indicaron referencias relevantes y actuales donde es posible encontrar más información.

En el capítulo 2 se presentó el objeto de estudio de este trabajo: *FANS*, un método de búsqueda por entornos o de búsqueda local con dos elementos novedosos: la utilización de una valoración difusa para evaluar las soluciones, y la utilización de varios operadores en el proceso de búsqueda.

Se mostró como definiciones particulares para la valoración difusa, en conjunción con el administrador de vecindario, permiten obtener o reflejar el comportamiento cualitativo de otros métodos de búsqueda por entornos, lo cual nos llevó a plantear que *FANS* puede considerarse como un framework o molde de heurísticas basadas en búsqueda local.

Teniendo en cuenta que para cada problema o instancia de problema un algoritmo puede funcionar mejor que otro (aunque no existen herramientas para determinar "a priori" cuál), creemos que la posibilidad de disponer de un sistema que pueda variar su comportamiento en forma sencilla, resulta de relevancia en el contexto de los sistemas de ayuda a la decisión.

Respecto a la utilización de varios operadores, la idea de "landscape" permitió establecer una justificación "intuitiva", pero convincente, de sus beneficios. Los experimentos realizados permitieron mostrar como, aún bajo un esquema muy simple de optimización (un hillclimber con multiarranque aleatorio), el uso de varios operadores permite obtener menores valores de error que los reportados cuando se utiliza un único operador.

En general, es conocido que existen tres alternativas para mejorar el comportamiento de un método de búsqueda local: 1) recomenzar la búsqueda desde otras soluciones, 2) aceptar transiciones a soluciones peores y/o 3) incorporar memoria para evitar ciclos y dirigir la búsqueda hacia zonas diferentes del espacio de búsqueda. Resulta al menos curioso que la utilización de varios operadores no haya sido considerada como una alternativa válida. Nuestra propuesta, en conjunto con una variante de VNS denominada *Variable Neighborhood Descent*, es un paso prometedor en dicha dirección.

En el capítulo 3 se verificó experimentalmente la utilidad de *FANS* como herramienta de optimización. Bajo las hipótesis de mínimo conocimiento del problema y número limitado de recursos (en este caso, cantidad de evaluaciones de la función de costo y/o cantidad de soluciones generadas), *FANS* fue capaz de obtener mejores o iguales resultados que varias versiones de algoritmos genéticos y recocido simulado sobre un número razonable (a nuestro entender) de instancias de tres problemas de optimización. Además se prestó especial cuidado en utilizar la misma plataforma computacional para realizar todos los experimentos y en casi todos los casos, se utilizaron algoritmos que fueron implementados por el autor de esta memoria. Los resultados obtenidos nos permiten concluir que *FANS* es útil en al menos, dos sentidos: primero, que *FANS* es una herramienta capaz de obtener soluciones razonablemente buenas y con poco esfuerzo computacional, y segundo, que dadas su simplicidad y

buenos resultados, *FANS* resulta muy útil para establecer líneas de base para la comparación con otros algoritmos más sofisticados.

Creemos que las hipótesis de experimentación planteadas son realistas, en el sentido que reflejan un escenario donde un decisor intenta obtener soluciones iniciales razonables para un problema nuevo. La flexibilidad de esta nueva heurística, permitirá incorporar gradualmente el conocimiento específico del dominio del problema a través de la utilización de definiciones más sofisticadas para los componentes de *FANS*, derivando en la obtención de mejores soluciones.

En el capítulo 4 comenzamos con algunos conceptos esenciales como gen, cromosoma, genoma, etc y describimos brevemente el Proyecto Genoma Humano para situarnos en contexto y presentar las nociones básicas de lo que a nuestro entender es la Bioinformática. Se describieron dos problemas centrales de la era post-genómica: el problema de predicción de estructura de proteínas y el problema de emparejamiento estructural de moléculas. En ambos casos se destacó que resultan interesantes en, al menos, dos aspectos: primero, las modelizaciones de los mismos dan lugar a problemas NP-Completos, con lo cual son un campo de prueba interesante para el desarrollo y prueba de técnicas metaheurísticas y, segundo, las soluciones que se obtengan tienen un impacto directo en el área de la biología molecular.

Posteriormente se describió la aplicación de nuestro algoritmo al problema de predicción de estructuras (*PSP*) sobre modelos en reticulados. En primer lugar, evaluamos la influencia que tenían dos esquemas de codificación basados en coordenadas internas (relativas vs. absolutas) sobre el comportamiento de algoritmos genéticos y *FANS*. Los resultados obtenidos concuerdan con los reportados en [76]: los algoritmos que utilizan codificación relativa sobre reticulado cuadrado, obtienen mejores resultados que aquellos que utilizan la absoluta; las diferencias se reducen sobre el reticulado cúbico y se detecta una ligera diferencia a favor de la codificación absoluta en el reticulado triangular. Por lo tanto, podemos concluir que si otras heurísticas van a aplicarse a esta versión de *PSP*, entonces estos resultados deben ser tenidos en cuenta a la hora de decidir la codificación a utilizar.

Como resultado adicional, estos experimentos nos permitieron detectar cuál es el mejor esquema para futuras aplicaciones de *FANS* sobre *PSP*: un esquema utilizando codificación relativa, en conjunto con un régimen de búsqueda donde se permitan transiciones a soluciones peores ($\lambda < 1$). Es interesante notar que este hecho también fue resaltado para algoritmos meméticos [81], donde se vió que las etapas de búsqueda local resultaban más productivas si permitían "empeorar" las soluciones.

En segundo lugar, se describió como la aplicación de algoritmos genéticos con operadores de cruce estandar para *PSP* presentaban un problema "teórico". La principal conclusión de dichos trabajos fué que los mismos resultados que obtenía el *AG* podían alcanzarse eliminando la población y mediante la aplicación de mutaciones sobre un único individuo. La aplicación de un esquema simple de *FANS* a *PSP* nos permitió obtener resultados mejores o iguales que los del *AG* sobre un conjunto de instancias de prueba, dando evidencia experimental para verificar la hipótesis que la población puede ser descartada si el *AG* utiliza un operador de cruce estandar y una representación basada en coordenadas internas.

En nuestra opinión, los dos aspectos analizados y los resultados obtenidos son relevantes de cara al diseño de mejores heurísticas para este problema.

La última tarea realizada fué mostrar la aplicación de *FANS* al problema de emparejamiento estructural de moléculas. Dado que el problema de comparación de estructuras no tiene una única respuesta y, más aún, la definición de una medida de similaridad entre proteínas está lejos de ser resuelta, la evaluación de *FANS* se hizo sobre un conjunto de instancias artificiales.

Se analizaron dos factores importantes que podían afectar la performance de *FANS* sobre este problema: la influencia del tamaño del patrón, y la influencia del parámetro λ utilizado para determinar la aceptabilidad de las soluciones.

El análisis del primer factor, reveló que, en promedio, el algoritmo fue capaz de obtener mejores soluciones para patrones grandes que para pequeños. Teniendo en cuenta el número de posibles soluciones en cada caso, los resultados pueden considerarse como una prueba que: 1) determinar el subconjunto

correcto de átomos es más complejo que determinar la permutación correcta de los mismos y 2) que *FANS* es capaz de resolver problemas con espacios de búsqueda de gran tamaño.

Acerca de la influencia del parámetro λ, no se pudo obtener evidencia suficiente para sugerir algún valor específico para futuras aplicaciones del algoritmo. Sin embargo, analizando los resultados en términos de optimización, podemos considerarlos como muy prometedores: para cada patrón, siempre existió un valor de λ, induciendo un comportamiento de *FANS*, que permitió obtener una solución exitosa en, a lo sumo, tres ejecuciones del algoritmo. Es decir, viendo nuevamente *FANS* como un framework, se pudo comprobar que el conjunto de los comportamientos inducidos por la variación de un único parámetro, resultó suficiente para resolver satisfactoriamente cualquier instancia del conjunto de prueba utilizado.

En resumen, consideramos que las principales aportaciones han sido:

- Proponer *FANS*, un nuevo método basado en búsqueda por entornos, adaptativo y difuso para problemas de optimización. La utilidad de sus dos elementos novedosos fué demostrada.

- Mostrar que *FANS* es una herramienta útil de optimización y relevante de cara a su incorporación en un sistema de ayuda a la decisión por su capacidad para capturar bajo un mismo esquema, el comportamiento cualitativo de una variedad de métodos de búsqueda local.

- Afirmar que frente a un problema nuevo, *FANS* es, en virtud de los resultados obtenidos, su simplicidad y flexibilidad, una herramienta a tener en cuenta para establecer valores de base para la comparación y/o validación de métodos más sofisticados o complejos.

- Mostrar la utilidad de *FANS* en el área de la Bioinformática a través de dos ejemplos:

  1.- el problema de predicción de estructuras, sobre el cual pudimos:

a.- confirmar que la influencia de la codificación en los resultados es similar a la detectada para algoritmos genéticos, estableciendo entonces, una guía para el desarrollo de otras heurísticas para este problema.

b.- dar soporte experimental a la hipótesis que no es necesario el uso de una población en un AG para PSP si este utiliza coordenadas internas y operadores de cruce estandar.

2.- el problema de emparejamiento estructural, donde sobre un conjunto de instancias artificiales, siempre se pudo obtener una versión de $FANS$ (inducida por el parámetro λ) que permitió resolver satisfactoriamente cada instancia con esfuerzo reducido.

Trabajo Futuro

A partir del trabajo presentado en esta memoria surgen tres líneas de investigación relacionadas que comentaremos a continuación. La primera está orientada al estudio de $FANS$ en sí misma, es decir a analizar otras definiciones para sus componentes y a profundizar en el impacto que cada uno de ellos tiene en el comportamiento global del algoritmo. La segunda está orientada a la aplicación de $FANS$ al área de la Bioinformática, con énfasis en modelos más complejos de PSP, y la evaluación de $FANS$ sobre casos reales del problema de emparejamiento estructural de moléculas. Finalmente la tercera, está orientada a utilizar $FANS$ para definir las etapas de búsqueda local en un algoritmo multimemético. En las secciones siguientes se detallan con más extensión las ideas en cada una de las líneas.

FANS como Método de Optimización

Bajo esta línea se pretende analizar con más profundidad la influencia que tienen diferentes definiciones para los componentes del algoritmo.

Todas las definiciones utilizadas en este trabajo, destacaban por su simplicidad. Sin embargo, también es necesario analizar el uso de definiciones más sofisticadas, por ejemplo, para el operador y el administrador de operación.

En principio, sabemos que el uso de esquemas simples resulta beneficioso, pero creemos que otros esquemas pueden incrementar el beneficio sin incrementar en exceso la complejidad del algoritmo.

Por ejemplo, en los operadores basados en k-intercambios, los administradores utilizados definen un orden $\{k_1, k_2, \ldots, k_m\}$. Cada vez que el administrador de vecindario falla en retornar una solución con k_t, el valor se adapta a k_{t+1}. Sin embargo, y de la misma forma que ocurre en VNS, también es razonable que el administrador haga $k = k_1$ cada vez que se puede obtener una solución aceptable con k_i, $i > 1$. De esta manera, si el algoritmo falla con k_m entonces podemos estar practicamente seguros de la optimalidad "local" de la solución actual.

Otra posibilidad que se planteó pero no se utilizó en esta memoria, consiste en disponer de una familia de operadores de modificación $\{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_k\}$. Bajo esta situación, el administrador de operación podría definir el orden de aplicación de los mismos. Dado que cada operador implica un vecindario diferente, obtendremos una estructura que recuerda a la de VNS. Sin embargo, nuestro esquema es diferente al de VNS ya que *FANS* no utiliza un método explícito de búsqueda local ni requiere una métrica de distancia entre las soluciones como la necesaria en VNS.

En el caso del administrador de vecindario, la exploración del vecindario realizada en los ejemplos era basicamente aleatoria. La única excepción se dió en la aplicación de *FANS* a *MSM* cuando se realizaba una exploración sistemática del vecindario inducido por el operador de modificación con $k = 1$. Sin llegar a utilizar una exploración exhaustiva, creemos que la utilización de una exploración sistemática con cada operador puede reducir el número de evaluaciones necesarias para encontrar el mejor valor. Por otro lado, en el caso de utilizar nuevamente operadores controlados por un único parámetro, creemos que puede ser beneficioso determinar el máximo número de intentos disponibles en el administrador en función de dicho parámetro.

Otra área a explorar es la de utilizar funciones de pertenencia diferentes para "Aceptabilidad" o incluso, variar la definición a lo largo de la ejecución del algoritmo. Esto puede lograrse de forma relativamente simple mediante

la utilización de modificadores lingüisticos sobre la valoración subjetiva. Por ejemplo, y utilizando la idea de Aceptabilidad, la búsqueda podría comenzar buscando soluciones *Algo Aceptables*, luego *Aceptables* y finalmente *Muy Aceptables*, logrando una transición suave entre las fases de exploración y explotación.

Un aspecto crucial en *FANS* y en todos los métodos heurísticos iterativos, es el relacionado con el criterio de parada. Existen criterios de parada difusos para algoritmos exactos [115] pero otros mecanismos son escasos. Una línea interesante a investigar, y no solo desde el punto de vista de *FANS*, es la que aparece en [108] donde se diseña una estrategia para detectar cuando una ejecución de recocido simulado es prometedora o no. Dado que este tipo de métodos usualmente se aplica más de una vez sobre la misma instancia, entonces una estrategia con ese objetivo puede resultar extremadamente útil para ahorrar tiempo y recursos.

FANS para *PSP*

Dentro de esta línea se pretende continuar con la aplicación de *FANS* sobre modelos en reticulados de *PSP* incrementando la longitud de las instancias del problema y analizando el comportamiento sobre el denominado modelo funcional de proteínas [58]. Este modelo es similar al modelo HP de Dill utilizado aquí, pero utiliza una función de energía diferente que provoca que las conformaciones óptimas presenten una "zona de encastre" que permitiría el acoplamiento de otras estructuras.

Dado que no existe acuerdo general sobre cual es *la mejor* función de energía para este problema, disponer de algoritmos que presenten un comportamiento robusto sobre varias de ellas, resulta fundamental para poder abordar la resolución de modelos más complejos del problema. En este aspecto, existen buenos resultados obtenidos mediante algoritmos multimeméticos [74], pero creemos que *FANS* puede obtener resultados similares con menor esfuerzo y mediante un esquema más simple.

En los experimentos realizados pudimos observar que existen instancias más "fáciles" que otras de resolver, por lo tanto otra línea de investigación

consiste en intentar caracterizar las instancias en términos de varios factores (por ejemplo, el tamaño, la relación entre cantidad de H's y P's, etc.) y determinar luego, para varios algoritmos, cuales resultan mas simples o complejas en términos de dichos factores. De esta manera podríamos obtener reglas del tipo: "para *FANS*, las instancias simples son las que tienen un relacion H/P alta y longitud media, y las complejas las que tienen H/P baja y long alta" o "para *AG*'s, resultan simples las que tienen cantidad similar de H's que de P's y long. media", etc; lo cual nos permitiría, dada una instancia o conjunto de instancias con ciertas características, concentrar los esfuerzos en un algoritmo o tipo de algoritmo en particular.

FANS para *MSM*

Respecto a la aplicación de *FANS* sobre el problema *MSM*, resulta claro que es necesario evaluar los resultados sobre un conjunto de instancias reales. La formulación del problema utilizada en esta memoria, es extremadamente general, es decir, en ningún momento se hace mención a que los datos son proteínas; simplemente, se utilizan dos conjuntos de puntos en el espacio y se desea buscar la "mejor" ocurrencia de uno (el patrón) en el otro (el target u objetivo).

Esto abre dos caminos que pueden llevar a la obtención de una versión más eficiente de *FANS*. El primero consiste en diseñar mecanismos que permitan reducir el espacio de búsqueda. Como ejemplo en esta línea podemos citar la incorporación al algoritmo de información específica que caracterice los conjuntos de puntos derivados de las proteínas (es decir, están lejos de ser arreglos aleatorios de puntos en el espacio). Por ejemplo los aminoacidos involucrados en una estructura de $\alpha-$hélice presentan una distribución en el espacio que es diferente a la correspondiente a las láminas$-\beta$. Se pretende utilizar este tipo de información para diseñar operadores de modificación que tengan en cuenta estas características. Otro aspecto interesante para reducir el espacio de búsqueda consiste en definir medidas de "compatibilidad" que permitan obtener para cada punto del patrón, una lista de posibles correspondencias en el objetivo. Así se reduciría considerablemente el espacio de soluciones del problema.

Otro enfoque consiste en ver que el problema en su forma más general, puede ser abordado con técnicas de reconocimiento de patrones[9] cuya aplicación puede dar pistas para encontrar soluciones a versiones más restringidas del problema; por ejemplo en aquellas donde los puntos representan los átomos de una molécula. Lo mismo puede decirse, como describimos en el Cap 4, de las modelizaciones basadas en grafos y detección de cliques. En cualquier caso, el problema a resolver resulta ser NP-Completo y por lo tanto creemos que versiones más sofisticadas de *FANS* pueden ser de utilidad para abordarlo.

FANS como Búsqueda Local en Algoritmos Multimemétic os

Los Algoritmos Meméticos son metaheurísticas diseñadas para encontrar soluciones a problemas de optimización complejos [91]. Basicamente, pueden entenderse como algoritmos evolutivos que incluyen una fase de optimización individual o "aprendizaje", como parte de la estrategia de búsqueda. Los Algoritmos Meméticos[10] también reciben el nombre de algoritmos genéticos híbridos, búsqueda local genética, etc. Un paso más allá de los algoritmos memético, son los denominados *Algoritmos Multimemétic os* (*MMA*)[79]. Un *MMA* puede ser visto como un algoritmo memético donde existen varios tipos de procedimientos de búsqueda local, denominados *memes*, disponibles para aplicar en la etapa de optimización local. En el contexto de un *MMA* los individuos están compuestos por una parte *genética*, que representa una solución al problema tratado; y una parte *memética* que codifica un meme (o procedimiento de búsqueda local) el cual será utilizado en la etapa de búsqueda local.

El conjunto de memes disponibles para los individuos, se denomina *memepool* y su diseño es un aspecto crítico para el éxito de la metaheurística [73]. Ya que *FANS* permite obtener una variedad de comportamientos algorítmicos de forma simple, compacta y elegante, se propone construir el memepool utilizan-

[9] De hecho, el nombre en inglés de la versión más general es "Point Set Pattern Matching", mirar por ejemplo [20, 49]

[10] El Dr. Pablo Moscato mantiene una página con amplia información sobre algoritmos meméticos, accesible en la dirección `htt://www.densis.fee.unicamp.br/~moscato/memetic_home.html`

do versiones simplificadas de *FANS*, para obtener una amplia gama de *memes difusos*. Creemos que una ventaja de esta modelización es que el ajuste de los memes resulta más simple cuando los memes están basados en *FANS*, que cuando estos son adaptativos como en [81, 73]. Mas aún, los memes basados en *FANS* admiten la incorporación de forma simple de conocimiento experto o información específica del problema.

Para realizar la administración de los memes (otro aspecto crucial de los *MMA*) se puede utilizar el esquema de herencia simple presentado en [79]. Este mecanismo permite que los memes más exitosos (asociados a individuos exitosos) se propaguen en la población mientras que aquellos que no reportan beneficios, sean eliminados automaticamente.

Esta variación de los memes se puede analizar a través del *valor de concentración* de los memes, que basicamente indica cuantos individuos de la población poseen cada meme. La representación gráfica de estos valores a través del tiempo permite mostrar claramente como en distintas etapas de la búsqueda, hay memes que se propagan en la población y otros que casi desaparecen de la misma dando lugar a un proceso de detección automática de las mejores estrategias o memes [73]. En cierta manera, esto evita realizar comparaciones simultáneas de diferentes estrategias; "simplemente" en un *MMA* se incluyen las estrategias disponibles (en este caso implementadas como versiones simplificadas de *FANS*) y el mecanismo de administración de los memes se encargará de dar relevancia a las prometedoras.

En la actualidad, ya hemos aplicado este esquema sobre *PSP* y podemos concluir que los resultados obtenidos son muy prometedores [80].

Bibliografía

[1] Emile Aarts and Jan Karel Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[2] R. Agarwala, S. Batzoglou, V. Dancik, S.E. Decatur, M. Farach, S. Hannenhalli, S. Muthukrishnan, and S.S. Skiena. Local rules for protein folding on a triangular lattice and generalized hydrophobicity in the hp model. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*. acm PRESS, 1997.

[3] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[4] Alexandre A. Andreatta, Sergio E. R. Carvalho, and Celso C. Ribeiro. An object oriented framework for local search heuristics. In *Proceedings of the 26th Conference on Technology of Object-Oriented Languages and Systems (TOOLS USA 98)*, pages 33–45. IEEE, 1998.

[5] C. Anfinsen, E. Haber, and et. al. The kinetics of formation of native ribonuclease during oxidation of the reduced polypeptide chain. In *Proceedings of the National Academy of Science USA*, volume 47, pages 1309–1314, 1961.

[6] Jonathan Atkins and William E. Hart. On the intractability of protein folding with a finite alphabet of amino acids. *Algorithmica*, pages 279–294, 1999.

[7] T. Bäck, D. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[8] Thomas Bäck. Genesys 1.0 user guide. Technical report, Univ. of Dortmund, Germany, 1992.

[9] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford Univ. Press, 1996.

[10] M.T. Barakat and P.M. Dean. Molecular structure matching by simulated annealing. I. A comparison between different cooling schedules. *Journal of Computer-Aided Molecular Design*, 4:295–316, 1990.

[11] M.T. Barakat and P.M. Dean. Molecular structure matching by simulated annealing. II. An exploration of the evolution of configuration landscape problems. *Journal of Computer-Aided Molecular Design*, 4:317–330, 1990.

[12] M.T. Barakat and P.M. Dean. Molecular structure matching by simulated annealing. III. The incorporation of null correspondences into the matching problem. *Journal of Computer-Aided Molecular Design*, 5:107–117, 1991.

[13] R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of heuristics*, 1(1):9–32, 1995.

[14] H.G Barrow and R.M Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters*, 4(83), 1976.

[15] J.E. Beasley. The or-library: a collection of test data sets. Technical report, Management School, Imperial College, London SW7 2AZ., 1997. http://mscmga.ms.ic.ac.uk/info.html.

[16] B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.

[17] A. Blanco, D. Pelta, and J.L. Verdegay. A fuzzy valuation-based local search framework for combinatorial problems. *Journal of Fuzzy Optimization and Decision Making*, 1(2):177–193, 2002.

[18] Armando Blanco, David Pelta, and Jose Luis Verdegay. Un método de búsquedas por entornos para optimización de funciones reales. In *X Congreso Español sobre Tecnologías y Lógica Fuzzy, Estylf 2000*, pages 423–428, 2000.

[19] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. Technical Report TR/IRIDIA/2001-13, IRIDIA: Institut de Recherches Interdisciplinaires et de Developpements en Intelligence Artificielle, Octubre 2001.

[20] Laurence Boxer. Faster point set pattern matching in 3d. *Pattern Recognition Letters*, 19:1235–1240, 1998.

[21] C. Branden and J. Tooze. *Introduction to Protein Structure*. Garland, New York, 1998.

[22] C. Bron and J. Kerbosch. Algorithm 457, finding all cliques of an undirected graph. *Communications of the ACM*, 16(575), 1973.

[23] M.J. Brusco and L.W. Jacobs. A simulated annealing approach to the solution of flexible labour scheduling problems. *Journal of the Operational Research Society*, 44(12):1191–1200, 1993.

[24] R. Carraghan and P.M. Pardalos. Exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375, 1990.

[25] S.C. Chan, A.K.C. Wong, and D.K.Y. Chiu. A survey of multiple secuence comparison methods. *Bulletin of Mathematical Biology*, 54:563–598, 1992.

[26] Colosimo, Montanari, and Sirabella. The application of a genetic algorithm to the protein folding problem. In *Proceedings of Engineering of Intelligent Systems (EIS 98)*, 1998.

[27] O. Cordón, F. Herrera, and M. Lozano. A classified review on the combination fuzzy logic-genetic algorithms bibliography. Technical Report DECSAI-95129, Dept. of Computer Science and A.I., University of Granada, 1995. Ultima Actualización en Diciembre de 1996, 544 referencias, http://decsai.ugr.es/h̃errera/fl-ga.html.

[28] David Corne, Marco Dorigo, and Fred Glover, editors. *New Ideas in Optimization*. McGraw-Hill, 1999.

[29] Earl Cox. *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems*. Academic Press, Boston, 1994.

[30] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):409–422, 1998.

[31] P. Crescenzi and V. Kann. A compendium of np optimization problems. Technical report, http://www.nada.kth.se/theory/problemlist.html.

[32] M. Delgado, J. Kacprzyk, J.L. Verdegay, and M.A. Vila, editors. *Fuzzy Optimization. Recent Advances*. Physica Verlag, 1994.

[33] A. Diaz, F. Glover, H. Ghaziri, J. Gonzalez, M. Laguna, P. Moscato, and F. Tseng. *Optimización Heurística y Redes Neuronales*. Ed. Paraninfo, 1996.

[34] Ken A. Dill. Dominant forces in protein folding. *Biochemistry*, 24:1501, 1985.

[35] E.I.Shakhnovich. Modeling protein folding: the beauty and power of simplicity. *Folding & Design*, 1:R50–R54, 1996.

[36] L. Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. J. E. Rawlings, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1991.

[37] E.S. Lander et.al. Initial sequencing and analysis of the human genome the genome international sequencing consortium. *Nature*, (409):860–921, 2001.

[38] J.C. Venter et.al. The sequence of the human genome. *Science*, 291(5507), 2001. Special issue on The Human Genome.

[39] Andreas Fink and Stefan Vos. Reusable metaheuristic software components and their application via software generators. In *Proceedings of the 4th Metaheuristics International Conference (MIC'2001)*, pages 637–641, 2001.

[40] Aviezri S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 6, 1993.

[41] Eleanor J. Gardiner, Peter J. Artymiuk, and Peter Willet. Clique-detection algorithms for matching three-dimensional molecular structures. *Journal of Molecular Graphics and Modelling*, 15:245–253, 1997.

[42] Michael Garey and David Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H . Freeman and Company, 1979.

[43] L. Di Gaspero and A. Schaerf. Easylocal++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. In *Proceedings of the 4th Metaheuristics International Conference (MIC'2001)*, 2001. Available at: http://tabu.dimi.uniud.it/EasyLocal.

[44] Genomics and its impact on medicine and society, a 2001 primer. Technical report, U.S. Department of Energy. Human Genome Program, 2001. Available on-line at: www.ornl.gov/hgmis.

[45] Fred Glover. Tabu search Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[46] Fred Glover. Tabu search Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[47] Fred Glover and Gary Kochenberger, editors. *Handbook on Metaheuristics*. Kluwer Academic Publishers, 2002. To appear.

[48] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, London, 1997.

[49] Steven Gold, Anand Rangarajan, Chieng-Ping Lu, Sugunna Pappu, and Eric Mjolness. New algorithms for 2d and 3d point matching: pose estimation and correspondence. *Pattern Recognition*, 31(8):1019–1031, 1998.

[50] D. Goldman, S. Istrail, and C. Papadimitriou. Algorithmic aspects of protein structure similarity. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 512–522, 1999.

[51] Garrison W. Greenwood, Jae-Min Shin, Byungkook Lee, and Gary B. Fogel. A survey of recent work on evolutionary approaches for the protein folding problem. In *Proceedings of the 1999 Congress on Evolutionary Computation CEC99*, pages 488–495. IEEE, 1999.

[52] Dan Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.

[53] Said Hanafi. On the convergence of tabu search. *Journal of Heuristics*, (7):47–58, 2000.

[54] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search Procedures for Optimization*, pages 433–458. Kluwer, 1999.

[55] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[56] Francisco Herrera and Manuel Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63, 2000.

[57] D. Higgins and W. Taylor (Eds.). *Bioinformatics: Sequence, structure and Databanks*. Oxford University Press, 2000.

[58] J.D. Hirst. The evolutionary landscape of functional model proteins. *Protein Engineering*, 12:721–726, 1999.

[59] L. Holm and C. Sander. Protein structure comparison by alignament of distance matrices. *Journal of Molecular Biology*, pages 123–138, 1993.

[60] Liisa Holm and Chris Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[61] J. Hooker. Needed: an empirical science of algorithms. *Operations Research*, 2(42):201–212, 1994.

[62] J. Hooker. Testing heuristics: We have it all wrong. *Journal of heuristics*, 1(1), 1995.

[63] 8th international fuzzy systems associations world conference, ifsa 99. Taipei, Taiwan, 1999.

[64] L. Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29–57, 1993.

[65] S. Istrail and W.E. Hart. Lattice and off-lattice side chain models of protein folding: Linear time structure prediction better than 86 In *Proceedings of the First International Conference on Computational Molecular Biology*, pages 137–146, 1997.

[66] Terry Jones. Crossover, macromutation, and population based search. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kauffman, 1995.

[67] Terry Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, 1995.

[68] Terry Jones. One operator, one landscape. In *Proc. Of the Twelfth Conference on Machine Learning*, 1995.

[69] Mehul Kimasia and Peter Coveney. Protein structure prediction as a hard optimization problem: The genetic algorithm approach. In *Molecular Simulations*, pages 47–55, 1997.

[70] R.K. Kincaid. A molecular structure matching problem. *Computers Ops Res.*, pages 25–35, 1997.

[71] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimisation by simulated annealing. *Science*, 220(4598):671–680, 1983.

[72] Patrice Koehl. Protein structure similarities. *Current Opinion in Structural Biology*, 11:348–353, 2001.

[73] N. Krasnogor. *Studies on the Theory and Design Space of Memetic Algorithms*. Ph.D. Thesis, Faculty of Computing, Mathematics and Engineering, University of the West of England, Bristol, United Kingdom., 2002.

[74] N. Krasnogor, B.P Blackburne, E. Burke, and J. Hirst. Multimeme algorithms for protein structure prediction. In *Proceedings of Parallel Problem Solving from Nature, PPSN'02*, 2002.

[75] N. Krasnogor, E. de la Canal, D. Pelta, D. H. Marcos, and W. A. Risi. Encoding and crossover mismatch in a molecular design problem. In Peter Bentley, editor, *Workshop 1 Notes of the Fifth International Conference on Artificial Intelligence in Design (AID'98)*, pages 22–27, 1998.

[76] N. Krasnogor, W.E. Hart, J. Smith, and D.A. Pelta. Protein structure prediction with evolutionary algorithms. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakaiela, and R.E. Smith, editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1596–1601. Morgan Kaufman, 1999.

[77] N. Krasnogor, D. Pelta, P. Martinez Lopez, P. Mocciola, and E. de la Canal. Genetic algorithms for the protein folding problem: A critical view. In C. Fyfe E. Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems, EIS'98*, pages 353–360. ICSC Academic Press, 1998.

[78] N. Krasnogor and J.E. Smith. Mafra: A java memetic algorithms framework. In Annie Wu, editor, *Workshop Program,Proceedings of the 2000 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2000. Available at: http://dirac.chem.nott.ac.uk/ñatk/Public/index.html.

[79] N. Krasnogor and J.E. Smith. Emergence of profitable search strategies based on a simple inheritance mechanism. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 2001.

[80] Natalio Krasnogor and David Pelta. *Fuzzy Sets based Heuristics for Optimization*, chapter Fuzzy Memes in Multimeme Algorithms: a Fuzzy-Evolutionary Hybrid. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2003. to appear.

[81] Natalio Krasnogor and Jim Smith. A memetic algorithm with self-adaptive local search: Tsp as a case study. In W. Banzhaf, J. Daida, M. Jakaiela, and R. Smith, editors, *GECCO-2000*. Morgan Kauffman, 2000.

[82] Giuseppe Lancia, Robert Carr, Brian Walenz, and Sorin Istrail. 101 optimal PDB structure alignments: A branch and cut algorithm for the maximum contact map overlap problem. In *Proceedings of the Fifth In-*

ternational Conference on Computational Molecular Biology, RECOMB 2001, 2001.

[83] N. Leibowitz, Z. Fligerman, R.Ñussinov, and H. Wolfson. Multiple structural alignment and core detection by geometric hashing. In T. Lengauer Et.Al., editor, *Procs of 7th Intern. Conference on Intelligent Systems for Molecular Biology ISMB 99.*, pages 167–177. AAAI Press, 1999.

[84] Helena R. Lourenco, Olivier Martin, and Tomas Stützle. Iterated local search. In Fred Glover and Gary Kochenberger, editors, *Handbook on Metaheuristics*. Kluwer Academic Publishers, 2002. To appear. Preliminary version at http://www.intellektik.informatik.tu-darmstad.de/tom/pub.html.

[85] N.M. Luscombe, D. Greenbaum, and M. Gerstein. What is bioinformatics? a proposed definition and overview of the field. *Method Inform Med*, 40(4):346–358, 2001.

[86] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[87] A.C. May and M.S. Johnson. Improved genetic algorithm-based protein structure comparisons: Pairwise and multiple superpositions. *Protein Engineering*, 8:873–882, 1995.

[88] ACW May. Towards more meaningful hierarchical classification of aminoacids scoring matrices. *Protein Engineering*, 12:707–712, 1999.

[89] J. Meidanis and J. Setubal. *An introduction to Computational Molecular Biology*. Freeman NY, 1996.

[90] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1999.

[91] P. A. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report Caltech Concurrent Computation Program Report 826, Caltech, Caltech, Pasadena, California, 1989.

[92] J.T. Ngo and J. Marks. Computational complexity of a problem in molecular structure prediction. *Proteing Engineering*, 5:313–321, 1992.

[93] I.H. Osman and G. Laporte. Metaheuristics: a bibliography. *Annals of Operations Research*, (63):513–623, 1996.

[94] Panos M. Pardalos and Mauricio G. Resende, editors. *Hanbook of Applied Optimization*. Oxford University Press, 2002.

[95] Mike Paterson and Teresa Przytycka. On the complexity of string folding. In *Proceedings of the First International Conference on Computational Molecular Biology*, 1997.

[96] Arnold L. Patton, W. Punch, and E. Goodman. A standard ga approach to native protein conformation prediction. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 574–581. Morgan Kauffman, 1995.

[97] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Procs. of the National Academy of Sciences of the USA*, 85:2444–2448, 1988.

[98] David Pelta, Armando Blanco, and Jose L. Verdegay. Applying a fuzzy sets-based heuristic for the protein structure prediction problem. *International Journal of Intelligent Systems*, 17(7):629–643, 2002.

[99] David Pelta, Armando Blanco, and Jose Luis Verdegay. A fuzzy adaptive neighborhood search for function optimization. In *Fourth International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies, KES 2000*, volume 2, pages 594–597, 2000.

[100] David Pelta, Armando Blanco, and Jose Luis Verdegay. Introducing fans: a fuzzy adaptive neighborhood search. In *Eigth International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems, IPMU 2000*, volume 3, pages 1349–1355, 2000.

[101] David Pelta, Armando Blanco, and Jose Luis Verdegay. Learning about fans behaviour: Knapsack problems as a test case. In *International Conference in Fuzzy Logic and Technology, EUSFLAT'2001*, volume 1, pages 17–21, 2001.

[102] David Pelta, N. Krasnogor, Armando Blanco, and Jose Luis Verdegay. F.a.n.s. for the protein folding problem: Comparing encodings and search modes. In *Fourth International Metaheuristics Conference, MIC 2001*, pages 327–331, 2001.

[103] A. Piccolboni and G. Mauri. Protein structure prediction as a hard optimization problem: The genetic algorithm approach. In N. et al. Kasabov, editor, *Proceedings of ICONIP '97*. Springer, 1998.

[104] David Pissinger. *Algorithms for Knapsack Problems*. PhD thesis, Dept. of Computer Science,University of Copenhagen, Denmark, 1995.

[105] A. A. Rabow and H. A. Scheraga. Improved genetic algorithm for the protein folding problem by use of a cartesian combination operator. *Protein Science*, 5:1800–1815, 1996.

[106] V. J. Rayward-Smith, editor. *Applications of Modern Heuristic Methods*. Alfred Waller, 1995.

[107] V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors. *Modern Heuristic Search Methods*. John Wiley, 1996.

[108] Norman M. Sadeh and Sam R. Thangiah. Learning to recognize (un) promising simulated annealing runs: Efficient search procedures for job shop scheduling and vehicle routing. In *Meta-Heuristics. Theory and Applications*, pages 277–298. Kluwer, 1996.

[109] Joseph D. Szustakowsky and Zhiping Weng. Protein structure alignment using a genetic algorithm. *PROTEINS: Structure, Function, and Genetics*, 38:428–440, 2000.

[110] JD Thompson, F Plewniak, and O Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.

[111] R. Unger and J. Moult. Finding the Lowest Free Energy Conformation of a Protein is an NP–hard Problem: Proof and Implications. Center for Advanced Research in Biotechnology. University of Maryland, 1993.

[112] R. Unger and J. Moult. A genetic algorithm for 3d protein folding simulations. In *Proceedings of the fifth Annual International Conference on Genetic Algorithms*, pages 581–588, 1993.

[113] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecular Biology*, 231(1):75–81, 1993.

[114] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[115] J.L. Verdegay and E. Vergara-Moreno. Fuzzy termination criteria in knapsack problem algorithms. *MathWare and Soft Computing*, 7(2-3):89–97, 2000.

[116] R. V. Valqui Vidal, editor. *Applied Simulated Annealing.* Springer-Verlag, 1993.

[117] S. Voss, S. Martello, I.H. Osman, and C, editors. *Meta-Heuristics. Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, 1999.

[118] Watson. *Molecular Biology of the Cell.* Freeman NY, 1996.

[119] R.Ñussinov H. Wolfson. Efficient detection of three dimensional structural motifs in biological macromolecules by computer vision techniques. *Procs. of the National Academy of Sciences U.S.A.*, 88:10495–10499, 1991.

[120] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[121] Lotfi A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, Part I. *Information Sciences*, (8):199–249, 1975. Part II, 8(1975), 301-357;Part III, 9(1975), 43-80.

[122] H. J. Zimmermann. *Fuzzy Sets Theory and Its Applications*. Kluwer Academic, 1996.