Herramientas de Soft Computing para la Comparación de Estructuras de Proteínas

Tesis Doctoral

Juan Ramón González González

Directores: David Alejandro Pelta
José Marcos Moreno Vega

Universidad de Granada
E.T.S Ingenierías Informática y de Telecomunicación
Departamento de Ciencias de la Computación e Inteligencia Artificial

Editor: Editorial de la Universidad de Granada
Autor: Juan Ramón González González
D.L.: Gr. 654 - 2008
ISBN: 978-84-338-4924-3

La memoria titulada **"Herramientas de Soft Computing para la Comparación de Estructuras de Proteínas"**, que presenta D. Juan Ramón González González para optar al grado de Doctor en Informática, ha sido realizada en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, bajo la dirección de los doctores D. David Alejandro Pelta, del mismo departamento, y D. José Marcos Moreno Vega, del Departamento de Estadística, Investigación Operativa y Computación de la Universidad de La Laguna.

Granada, Febrero de 2008

Juan Ramón González González

David Alejandro Pelta

José Marcos Moreno Vega

A quienes llenan mi corazón de emociones, mi mente de ideas...

Índice general

Agradecimientos **17**

Introducción **19**

1. Soft Computing y Sistemas de Ayuda a la Decisión **25**
1.1. Soft Computing 25
1.1.1. Modelos probabilísticos 33
1.1.2. Lógica difusa 35
1.1.3. Redes neuronales 37
1.1.4. Metaheurísticas 40
1.2. Sistemas de Ayuda a la Decisión 57
1.3. Resumen 62

2. Bioinformática **65**
2.1. Fundamentos de la estructura de proteínas 68
2.1.1. Estructura primaria de las proteínas: la secuencia de aminoácidos 69
2.1.2. Estructura secundaria de las proteínas 71
2.1.3. Estructura terciaria de las proteínas: La estructura tridimensional global 75
2.1.4. Estructura cuaternaria de las proteínas: Asociaciones de múltiples cadenas polipéptidas 78
2.1.5. Bases de datos de proteínas: PDB, CATH y SCOP . . 78
2.2. Comparación de estructuras de proteínas 83
2.2.1. Consideraciones Generales 83
2.2.2. Comparación de estructuras vía Matrices de Distancia usando DALI 86

2.2.3. Comparación de estructuras vía Matrices de Distancia usando MatAlign 87
2.2.4. Comparación de Estructuras vía Cliques 87
2.2.5. Comparación de estructuras vía extensión combinatoria de la trayectoria óptima 89
2.2.6. Mapas de contacto 90
2.2.7. USM: Comparación de mapas de contactos mediante la medida universal de similaridad 95
2.2.8. Max-CMO: Problema de la máxima superposición de mapas de contacto 96
2.2.9. GMax-FCMO: Problema generalizado de la máxima superposición de mapas de contacto difusos 101
2.3. Conclusiones 102

3. SiGMA: Un esqueleto para Sistemas de Ayuda a la Decisión basados en optimización 105
3.1. Descripción, requisitos y diseño de SiGMA 107
3.1.1. Detalles técnicos 109
3.2. Trabajando con SiGMA 111
3.2.1. Administración de resolutores 113
3.2.2. Ejecución de resolutores 114
3.2.3. Análisis de resultados 117
3.3. Ejemplos de aplicación 118
3.3.1. Problema de localización de facilidades: el modelo del p-hub 119
3.3.2. Problema de comparación de estructuras de proteínas 122
3.4. Conclusiones 124

4. Resolución del modelo Max-CMO de comparación de estructuras de proteínas 127
4.1. Algoritmo MSVNS de Búsqueda por Entornos Variable Multiarranque 128
4.2. Experimentos 138
4.2.1. Análisis de Características Operativas del Receptor (ROC) 139
4.2.2. ¿Es beneficioso el uso de MSVNS desde el punto de vista de optimización? 143

4.2.3. ¿Puede proporcionar MSVNS un ranking de similaridad de proteínas apropiado? 154
4.3. Conclusiones 167

5. Comparación de proteínas con mapas de contacto difusos 171
5.1. Comparación de proteínas mediante mapas de contacto difusos y la medida universal de similaridad 171
5.1.1. Experimentos 172
5.2. Resolución del GMax-FCMO 176
5.2.1. Experimento 1: ¿Puede detectarse la similaridad entre proteínas usando GMax-FCMO? 177
5.2.2. Experimento 2: Relación entre los valores de superposición estándar y difusos 182
5.3. Conclusiones 187

Conclusiones 189

Trabajo futuro 193

Bibliografía 195

Índice de figuras

1.1. Componentes de la Soft Computing. 33
1.2. Ejemplo de función de pertenencia de un conjunto difuso. . . 36
1.3. Ejemplo de red neuronal multicapa con conexiones feedforward. 39

2.1. Estructura básica de un aminoácido. 70
2.2. Enlace peptídico. Los dos aminoácidos marcados en gris claro se unen mediante el enlace peptídico indicado en gris oscuro. Los grupos R representan cualquiera de las 20 posibles cadenas laterales de los aminoácidos. 71
2.3. Hélice α. 73
2.4. Láminas β paralelas (a y b) y antiparalelas (c y d). 74
2.5. Fragmento de fichero PDB para la proteína con código 1AA9. 80
2.6. Representación mediante matriz del mapa de contacto de la proteína 1AA9. 91
2.7. Representación mediante grafo del mapa de contacto de la proteína 1LYZ. 92
2.8. Representación mediante matriz del mapa de contacto difuso de la proteína 1AA9 para dos tipos de contactos: cortos y largos. 93
2.9. Tres conjuntos de funciones de pertenencia utilizables para crear mapas de contacto difusos. 94
2.10. Ejemplo de un alineamiento entre dos proteínas. El valor de superposición (valor objetivo de la solución) es 2 porque en la solución aparecen dos ciclos de longitud 4. El primer ciclo se compone de las aristas $(1 \in P_1, 3 \in P_1)$, $(3 \in P_1, 5 \in P_2)$, $(5 \in P_2, 2 \in P_2)$ y $(2 \in P_2, 1 \in P_1)$. El segundo ciclo tiene las siguientes 4 aristas: $(2 \in P_1, 6 \in P_1)$, $(6 \in P_1, 7 \in P_2)$, $(7 \in P_2, 3 \in P_2)$, y $(3 \in P_2, 2 \in P_1)$. 97

2.11. Ejemplo de un alineamiento entre dos mapas de contacto difusos. 102

3.1. Ventana principal de SiGMA. 112
3.2. Pestaña Solver Administration de SiGMA: Añadiendo un *resolutor*. 113
3.3. Pestaña Solver Execution de SiGMA: Ejecutando un algoritmo. 115
3.4. Pestaña Result Analysis de SiGMA: Visualización de los resultados de una ejecución. 118
3.5. Ejemplo de salida gráfica en SiGMAPhub. 121
3.6. Salida gráfica en SiGMAProt (usando Jmol). 123

4.1. Vector para almacenar una solución de ejemplo. 130
4.2. Ejemplo de la función *simplificar* - El *par* $3 \leftrightarrow 6$, formado por $6 \in P_1$ y $3 \in P_2$, será eliminado. 132
4.3. Ejemplo de la estructura de entorno *neighborhoodMove*. Se elige un *par* a mover y se identifican sus intervalos de movimiento factible (a). A continuación, se selecciona un *par* factible de la región delimitada por los intervalos anteriores y se sustituye con él el *par* original. 134
4.4. Ejemplo de la estructura de entorno *neighborhoodAdd*. Se elige el residuo $4 \in P_1$ para ser emparejado. El nuevo *par* creado puede ser factible, como se muestra en (a), o no factible (b). En este segundo caso, la factibilidad se restaura eliminando los *pares* $(2 \in P_1, 3 \in P_2)$ y $(3 \in P_1, 5 \in P_2)$. 136
4.5. Detalle del espacio de una curva ROC. 141
4.6. Resultados de superposición para proteínas alpha-beta. Los valores de LGA y SGMA han sido tomados de la tabla 5 de [76]. 146
4.7. Resultados de superposición en un conjunto mixto de proteínas. Los valores de LGA y SGMA han sido tomados de la tabla 6 de [76]. 147
4.8. Distribución del error (gap) sobre los valores exactos (en %) para el conjunto de instancias no resueltas de forma óptima por MSVNS en el conjunto de datos de Lancia. 151
4.9. Distribución del error (gap) sobre los valores exactos (en %) para el conjunto de instancias no resueltas de forma óptima por MSVNS en el conjunto de datos de Skolnick. 152

4.10. Agrupamiento jerárquico basado en los valores de superposición normalizados (usando $Norm1$) de todos los pares de proteínas en el conjunto de datos de Skolnick. Los dendongramas superiores (a, b) corresponden a agrupamiento por enlace simple y los inferiores (c, d) a agrupamiento por enlace promedio. Por otro lado, los dendogramas a la izquierda (a, c) corresponden a valores provenientes de MSVNS1 mientras que los dendogramas a la derecha (c, d) corresponden a resultados de MSVNS3. 157
4.11. Curva ROC para cada medida sobre el conjunto de datos de Fischer en la clasificación a nivel del plegamiento en SCOP. . 159
4.12. Valores de AUC para cada medida sobre el conjunto de datos de Fischer en la clasificación a nivel del plegamiento en SCOP diferenciando según los distintos plegamientos. 160
4.13. Valores de AUC para cada medida sobre el conjunto de datos de Fischer en la clasificación a nivel de la clase en SCOP, diferenciando los resultados en función de la clase de la proteína 1. 161
4.14. Curvas ROC para cada medida en la base de datos Nh3D para el nivel de arquitectura en CATH. 164
4.15. Examples of ROC curves to show how dependent the strategies are on the architecture type of the query. 165
4.16. Valores de AUC para cada medida y arquitectura de la proteína de consulta en el conjunto de datos Nh3D. Se ve claramente que ninguno de los métodos se sitúa por encima de los otros para todas las arquitecturas. 166

5.1. Definiciones de funciones de pertenencia usadas para crear los mapas de contacto difusos para USM. 173
5.2. Clusters obtenidos para las matrices de similaridad resultantes de la comparación mediante USM de los mapas de contacto difusos de las proteínas de Chew-Kedem. Las letras entre paréntesis se corresponden con las 6 definiciones de funciones de pertenencia usadas para crear los mapas de contacto difusos, que se mostraban en la figura 5.1. 175
5.3. Definiciones usadas para crear los mapas de contacto difusos para MSVNS. 179

5.4. *normFitness* medio para cada par de clases con cada definición (función). 181
5.5. Dispersión de las diferencias de costo entre crisp-8.0 y twoTrapezesBelow8 frente a la diferencia de contactos. 185
5.6. Dispersión de las diferencias de costo entre crisp-8.0 y twoTrapezesBelow8 frente a la diferencia de residuos. 186

Índice de tablas

3.1. Ejemplo de los valores de los parámetros durante la ejecución de un resolutor. Se asume que se ha lanzado el resolutor pidiendo 2 repeticiones y que como parámetros se tiene una semilla aleatoria (**SemAl**), un valor de tipo entero (**ent**) y dos parámetros de tipo *FILE_OF_VALUES* (**fv1** y **fv2**). El contenido de **fv1** se asume que son 3 líneas con los valores "1", "2" y "3", mientras que **fv2** contiene dos líneas con los valores "a" y "b". En la tabla se muestran todas las ejecuciones individuales producidas para esta configuración. 117

4.1. Matriz de confusión (tabla de contingencia) para los cuatros posibles resultados de un clasificador binario. 140

4.2. Información de los conjuntos de datos de Skolnick y Lancia. . 148

4.3. Resultados sobre los 2702 pares del conjunto de datos de Lancia. 150

4.4. Resultados sobre los 161 pares del conjunto de datos de Skolnick. 150

4.5. Tiempos totales empleados por cada una de las versiones de MSVNS para comparar los pares de proteínas con valores exactos disponibles (2702 pares en el conjunto de datos de Lancia y 161 en el de Skolnick). 153

4.6. Información del conjunto de datos de Fischer. 158

4.7. Valores de AUC para cada medida sobre el conjunto de datos de Fischer para la clasificación a nivel de plegamiento en SCOP. 159

4.8. Valores de AUC para cada medida sobre el conjunto de datos de Fischer a nivel de clase. 161

4.9. Información del conjunto de datos Nh3D. 163

4.10. Valores de AUC para cada medida sobre la base de datos Nh3D. El experimento consistió en el uso de 73 proteínas de consulta sobre 806 dominios, realizándose el análisis al nivel de arquitectura en CATH. 163
4.11. Tiempos de ejecución para MSVNS, DaliLite y MatAlign. Los tiempos corresponden a la comparación de 8 pares formados por las proteínas más grandes de la base de datos Nh3D. Todas las ejecuciones se realizaron en la misma máquina y puede verse cómo MSVNS es claramente más rápido que DaliLite. MatAlign mejora los tiempos de MSVNS pero a costa de proporcionar los peores valores para realizar la clasificación. . . . 168

5.1. Análisis de los tamaños de las proteínas en cada clase del conjunto de datos. 178

Agradecimientos

El periodo de tiempo en el que se ha desarrollado esta tesis ha marcado grandes cambios en toda mi vida, no sólo en el aspecto académico/laboral. La oportunidad de realizar la tesis en el seno del grupo MODO me vino en un momento de mi vida lleno de incertidumbre tras terminar la carrera y encontrarme un panorama laboral poco prometedor.

Mi idea inicial tras terminar mis estudios de ingeniería era la incorporación al mundo laboral privado, y no contemplé la opción del tercer ciclo hasta comprobar las escasas oportunidades laborales que se me ofrecían en Tenerife en aquella época, con lo que, amparado en una pequeña beca de una entidad de ahorros local, empecé a realizar allí un primer año de doctorado sin garantías del mantenimiento de la financiación y sin posibilidad de emancipación debido a la precariedad económica.

Fue entonces cuando un compañero de estudios me puso en contacto con Marcos Moreno, y a través de él me puse en contacto con José Luis Verdegay que me abrió la vía a venir a trabajar en Granada bajo el amparo económico de una beca de Formación de Personal Investigador (FPI). Esto me permitió realizar un cambio de aires en mi vida e incorporarme a un grupo de investigación que, además de compañeros de trabajo, han sido buenos amigos, y debo darles las gracias puesto que sin ellos, el trabajo que con esta memoria se culmina no hubiera sido posible. Gracias, por tanto, a Curro, Maite, David, Natalio, Alejandro, Carlos, Socorro, Antonio, Nacho y a varios otros investigadores y compañeros que no nombro por no hacer eterna la lista pero a quienes no olvido y a los que también debo consejos y ánimos que me ayudaron a este trabajo. Además, en lo que a investigadores y amigos se refiere, quiero dar gracias especialmente a mis directores Marcos Moreno y David Pelta.

Debo agradecer, igualmente, el soporte económico prestado por el Ministerio de Educación y Ciencia, a través de la beca FPI asociada al proyecto

HeuriFuzzy (TIC2002-04242-C03-02) y su posterior continuación en el proyecto HeuriCosc (TIN2005-08404-C04-01), así como a la posibilidad que me brindaron de realizar estancias de investigación en las universidades de La Laguna y de Nottingham durante los años 2005 y 2006 respectivamente.

Quiero dar las gracias también a los otros becarios de investigación, que me acogieron y llegaron a ser de mis primeros amigos al llegar a Granada. A Wane, que ella sí que fue la primera entre las primeras, ayudándome a conocer Granada e integrándome en su grupo de amigos cuando apenas conocía a nadie. Y quiero agradecer también a todos los demás amigos que han hecho más agradable mi estancia aquí, los del trabajo y los de fuera, a los que siguen cerca y a los que se han ido, a los que comparten piso conmigo y a los que no, a los que comparten las noches de fiesta y los que se toman solamente unas tapas o un café de vez en cuando. A todos ellos, gracias, pues os debo en buena medida mi evolución personal de los últimos años, que ha sido tan importante como la académica, o incluso más.

Y gracias, por último, a mi familia, por haber comprendido mis motivos para venirme a Granada y por recibirme siempre tan bien en las pocas visitas que puedo hacerles a lo largo del año.

A todos, un gran abrazo.

Introducción

En el año 2003, el Proyecto Genoma Humano consiguió determinar las posiciones relativas de todos los pares de bases (nucleótidos) que componen el genoma humano, e identificar los más de 20000 genes presentes en él. Se trataba del logro de un proyecto muy ambicioso que había sido iniciado en 1990 por el Departamento de Energía y los Institutos de la Salud de los Estados Unidos, con un gran presupuesto de 3000 millones de dólares y un plazo de realización de 15 años, y que pudo cumplirse dos años antes gracias a los avances tecnológicos.

Lejos de suponer el final de la Genómica, los datos del Proyecto Genoma abrieron nuevas vías de investigación en los campos de la medicina y la biotecnología. La explosión de datos que ha supuesto el Proyecto Genoma, y todas las nuevas investigaciones, han dado origen a la disciplina de la Bioinformática, que intenta servir como medio para el manejo de estos datos y la realización de experimentos.

La Bioinformática se encarga del desarrollo y aplicación de algoritmos y métodos para transformar datos en conocimiento del sistema biológico. Para ello han de proporcionarse herramientas que permitan cruzar información de secuencias, estructuras, microarrays, información textual, etc., de manera que la integración de esta información pueda conducir a la creación de conocimiento nuevo. Los problemas en los que se aplica la Bioinformática son variados: diseño de bases de datos, comparación y alineamiento de secuencias y de estructuras, construcción de árboles filogenéticos, ensamblado de fragmentos, mapeo físico de cromosomas, rearreglo de genomas, predicción de estructuras proteicas, análisis de perfiles de expresión génica en microarrays, etc. El rápido incremento en el número de estructuras macromoleculares 3D ha conducido a la aparición de una subdisciplina de la Bioinformática: la Bioinformática estructural, que se ocupa de la representación, almacenamiento, recuperación, análisis y visualización de información

estructural a escala atómica y subcelular.

La Bioinformática estructural, como otras muchas subdisciplinas de la Bioinformática, se caracteriza por dos metas u objetivos: la creación de métodos de propósito general para la manipulación de información sobre macromoléculas biológicas, y la aplicación de estos métodos a la resolución de problemas en Biología y la creación de conocimiento nuevo. Estas dos metas están profundamente interconectadas, porque parte de la validación de los nuevos métodos consiste en comprobar su éxito al resolver problemas reales. Al mismo tiempo, los desafíos actuales en Biología demandan el desarrollo de nuevos métodos que puedan manejar el volumen de datos actualmente disponible y la complejidad de los modelos que los científicos tienen que crear para explicar estos datos.

Un problema de vital importancia dentro de la Bioinformática estructural es el problema de comparación de estructuras de proteínas. Su importancia radica en que las proteínas desempeñan funciones importantísimas para la vida, o pueden usarse en la fabricación de nuevos medicamentos, y la función concreta que desempeñan está íntimamente relacionada con su estructura. En este contexto, una forma rápida de comparar las proteínas nuevas (o artificiales) con las proteínas cuya función ya es conocida podría acelerar notablemente la investigación biológica.

En muchos de estos campos, las técnicas de Soft Computing pueden jugar un papel crucial, y de hecho lo juegan, al estar especialmente capacitadas para conseguir resultados de forma eficiente y con un buen nivel de calidad, además de permitir realizar modelizaciones que tengan en cuenta las imprecisiones o las carencias de información que pueden existir en los datos (debidas a conocimiento impreciso, a potenciales errores en las mediciones, etc.). En muchos casos, además, las soluciones óptimas de los modelos no son necesarias, pues puede haber mínimos (máximos) irrelevantes, y lo que interesa realmente es encontrar la solución biológicamente relevante.

Por otro lado, los Sistemas de Ayuda a la Decisión (SAD) son especialmente adecuados para el control centralizado de los diversos métodos existentes para resolver un mismo problema, y para el manejo de grandes volúmenes de datos, de forma que se puedan validar de forma experimental los algoritmos, constituyendo este comportamiento práctico una de sus características más relevantes. Tanto los SAD como las técnicas de Soft Computing pueden ayudar en la generación e integración de información para dar

lugar a conocimiento biológico nuevo.

Dada la importancia de la Bioinformática y, en concreto, del problema de comparación de estructuras de proteínas, así como la capacidad y la utilidad de la Soft Computing para abordar problemas complejos y de forma tolerante a la imprecisión; así como la capacidad de los SAD para facilitar el manejo de numerosos métodos y grandes volúmenes de datos, la tesis aquí propuesta se plantea los objetivos siguientes:

1. Proporcionar métodos eficientes y de calidad para la comparación de estructuras de proteínas, que complementen y/o mejoren los ya existentes y posibiliten el mejor desarrollo de otras investigaciones.

2. Utilizar la Soft Computing para la creación de los métodos anteriores, de forma que se puedan aprovechar sus cualidades para la creación de métodos mejores.

3. Facilitar herramientas y técnicas que permitan tratar con la información imprecisa propia del problema de comparación de proteínas y, en particular, profundizar en el uso de una modelización difusa del problema de comparación.

4. Comprobar que los métodos anteriores tienen utilidad desde el punto de vista biológico.

5. Crear un SAD que ayude en la experimentación asociada a los nuevos métodos, permitiendo manejar de forma sencilla todos los métodos y datos asociados al problema de comparación de proteínas.

Para llevar a cabo estas tareas y alcanzar los objetivos propuestos, la tesis se ha estructurado en 5 capítulos, que se describen a continuación. En el capítulo 1 se hace una descripción de lo que son la Soft Computing y los SAD, pues son estas las bases sobre las que se apoya todo el trabajo realizado. En lo referente a Soft Computing se hará una presentación de cuál es el estado actual del área, para enmarcar el contexto en que esta tesis ha sido desarrollada e indicar en qué medida afecta a la investigación específica en que se centra la tesis. La descripción de los SAD se hará, en cambio, poniendo énfasis en los aspectos necesarios para el desarrollo de nuestros sistemas.

El capítulo 2 entra de lleno en el campo de la Bioinformática, explicando sus orígenes y los problemas de los que se ocupa, viéndose luego en más

detalle el problema de comparación de estructuras de proteínas que se aborda en esta tesis. Para ello, se presentarán los aspectos básicos de Biología que son necesarios para la comprensión de este trabajo y se dará una visión global de la Bioinformática estructural. Veremos los diferentes niveles estructurales que existen, la importancia que tienen las proteínas y la relación que existe entre su estructura y su función. Se describirán también las bases de datos existentes con información tridimensional y de clasificación de proteínas, y diferentes modelos y métodos que se utilizan para realizar su comparación. Entre estos métodos y modelos se presentarán los que se utilizan en esta tesis: los mapas de contacto estándar y difusos, el uso de la medida universal de similaridad (USM) como medio para comparar mapas de contacto y dos modelos de comparación de proteínas orientados a dar un alineamiento o correspondencia entre los elementos (aminoácidos, residuos o átomos) de las proteínas comparadas, de forma que se maximice la superposición entre sus mapas de contacto.

La primera aportación de esta tesis se describe en el capítulo 3. En él se presenta SiGMA, un esqueleto para generar Sistemas de Ayuda a la Decisión basados en optimización que pretende servir de ayuda en todo tipo de problemas en los que exista un gran volumen de datos y de métodos de resolución (resolutores), como son los problemas de comparación de estructuras de proteínas que se presentan en la tesis. Pero SiGMA no es simplememente un SAD para un propósito específico, sino que procura ser un esqueleto genérico y extensible de forma dinámica, con capacidad para: facilitar la incorporación de resolutores preexistentes sin tener que reimplementarlos en ningún lenguaje específico; recolectar y mostrar al usuario cualquier resultado obtenido por los resolutores de forma sencilla y sistemática; permitir agregar, modificar y borrar resolutores de un modo dinámico, incluyendo la generación dinámica de las interfaces gráficas de usuario (GUI) necesarias para la ejecución de los resolutores y el análisis de sus resultados; y proporcionar una base de datos de resolutores que permita reutilizarlos o modificarlos en cualquier momento. Más aún, SiGMA está basado en herramientas de código abierto y diseñado de forma que puede extenderse fácilmente o portarse a distintas plataformas. Aparte de la descripción de SiGMA, se presentarán dos aplicaciones para la creación de dos SAD para problemas concretos: el problema del p-hub y el problema de comparación de estructuras de proteínas del que es objeto esta tesis.

Finalmente, los capítulos 4 y 5 presentan las herramientas de Soft Computing para el problema de comparación de proteínas. En primer lugar, en el capítulo 4 se describe un algoritmo heurístico de Búsqueda por Entornos Variables Multiarranque (MSVNS) para resolver la comparación de proteínas a través de la superposición de mapas de contacto estándar, estudiando su comportamiento tanto a nivel de valores de optimización obtenidos como, en términos biológicos, en cuanto a su capacidad de producir una ordenación (ranking) de similaridad adecuada en un conjunto de proteínas. Por último, el capítulo 5 muestra cómo realizar la comparación de proteínas a partir de mapas de contacto difusos tanto a través de USM como de una nueva versión del algoritmo heurístico MSVNS. Se estudia, en primer lugar, cómo el uso de mapas de contacto difusos puede presentar ventajas sobre los mapas de contacto estándar y, a continuación, se comprueba cómo la similaridad puede detectarse de manera correcta mediante la superposición de mapas de contacto difusos y se analiza la relación que existe entre los valores de superposición estándar y difusos.

Cada uno de los capítulos anteriores incorpora una sección de resumen o de conclusiones al final, y tras el capítulo 5 se presentan de forma global las conclusiones de toda la tesis y, como trabajo futuro, las ideas que han surgido para abrir nuevas líneas de investigación. La memoria termina con la relación de la bibliografía más relevante consultada para su preparación.

Capítulo 1

Soft Computing y Sistemas de Ayuda a la Decisión

En este capítulo se describen los elementos principales de la Soft Computing y los Sistemas de Ayuda a la Decisión (SAD), dado que serán utilizados posteriormente en la creación de los nuevos métodos de comparación de estructuras de proteínas y para la creación del SAD que servirá para su correcto manejo y la realización de la experimentación. Se intentará dar una visión del estado actual del arte en ambos campos además de resaltar todos los aspectos necesarios para el desarrollo de nuestras herramientas.

La descripción de la Soft Computing (sección 1.1) se basa en el trabajo de Verdegay, Bonissone y Yager [134], y se completa, en lo que a las componentes clásicas de la Soft Computing se refiere, con bibliografía adicional. Por otro lado, para la sección de SAD (1.2) hay un mayor número de puntos de vista sobre el área y su historia, y las referencias utilizadas han sido varias, estando convenientemente indicadas. En dicha sección se pone énfasis en los SAD basados en optimización, dado que serán necesarios para la herramienta que se presenta en el capítulo 3.

1.1. Soft Computing

Para entender la aparición de la Soft Computing hay que resaltar, en primer lugar, que la necesidad de encontrar la solución óptima de un problema correctamente planteado, o la mejor solución entre las disponibles, justifica que se construyan y estudien teorías, y se propongan metodologías

adecuadas al campo científico en el que surge la cuestión que se ha de resolver. Desde un punto de vista más concreto, pero aún muy general, una importante clase de problemas son los conocidos con el nombre de problemas de optimización, habitualmente asociados a tener que encontrar el máximo o el mínimo valor que una determinada función puede alcanzar en un cierto conjunto previamente especificado. Todo lo relativo a estos problemas se enmarca dentro del cuerpo doctrinal denominado Programación Matemática, que incluye una enorme variedad de situaciones, según que se consideren casos lineales, no lineales, aleatoriedad, un solo decisor o varios decisores, etc. Entre todos los modelos que se incluyen en la Programación Matemática, el más y mejor estudiado, así como el que ha probado tener unas repercusiones prácticas más importantes, es el correspondiente al caso lineal uni-objetivo, tema del que se ocupa la Programación Lineal. Los métodos y modelos de la Programación Lineal tienen relevantes aplicaciones en las diferentes áreas de las Ingenierías, la Economía, las Matemáticas, la Investigación Operativa, la Inteligencia Artificial, y demás disciplinas más o menos relacionadas con la optimización, y constituyen un sustrato teórico más que adecuado para abordar de un modo elegante y eficiente situaciones muy complejas.

Cuando en los problemas de Programación Matemática se consideran elementos de naturaleza borrosa, surgen los métodos de optimización borrosa, quizás una de las áreas mas fructífireas en el ámbito de lo fuzzy, tanto desde el punto de vista teórico como aplicado, que, aunque recoge métodos y modelos que dan solución a una enorme variedad de situaciones prácticas reales, del mismo modo que le ocurre a la Programación Matemática convencional, no puede dar respuestas en todos los escenarios posibles. Y no puede hacerlo por una sencilla razón, y es que hay problemas que siendo planteables en términos propios de ese campo, no son resolubles con sus técnicas.

La facilidad de resolver problemas reales de dimensión cada vez mayor, gracias a la mayor potencia y el menor costo de los computadores, la imposibilidad de conocer en todos los casos las soluciones exactas que les corresponden a esos problemas, y la necesidad de dar respuestas a las situaciones prácticas contempladas en multitud de casos (problemas de organización de las tareas que ha de efectuar un robot, de identificación de itinerarios, de clasificación y ubicación de recursos, de recorte,...; en definitiva, problemas combinatorios), han motivado que los algoritmos de tipo heurístico sean

empleados cada vez más como valiosas herramientas capaces de proporcionar soluciones donde los algoritmos exactos no son capaces de encontrarlas. Así en los últimos años ha surgido un largo catálogo de técnicas diversas, animadas por el principio de que es mejor satisfacer que optimizar, o, lo que es lo mismo, que antes de no poder dar la solución óptima a un problema, es mejor dar una solución que satisfaga al usuario en algún sentido que previamente habrá especificado. Estas técnicas se han demostrado extraordinariamente efectivas, y algunos ejemplos de ellas pueden ser los algoritmos de Búsqueda Tabú, Enfriamiento Simulado, GRASP (Greedy Randomized Adaptive Search Procedure), Algoritmos Genéticos, u otras más recientes: Algoritmos Meméticos, VNS (Búsqueda por Entornos Variables), Colonias de Hormigas, Estimación de Distribuciones, Búsqueda Dispersa, Programación por Restricciones,... Todo este listado muestra el gran interés de este campo, y la falta de un marco teórico en el que encuadrar, relacionar y poder comparar estos algoritmos.

Desde que en 1965 Lotfi A. Zadeh [142] introdujera el concepto de conjunto difuso (fuzzy set) permitiendo la pertenencia de un elemento a un conjunto de forma gradual, y no de manera absoluta como establece la teoría conjuntista clásica, es decir, admitiendo pertenencias valoradas en el intervalo $[0, 1]$ en lugar de en el conjunto $\{0, 1\}$, las aplicaciones y desarrollos basados en este sencillo concepto han evolucionado de tal modo que, hoy en día, es prácticamente imposible calcular el volumen de negocio que generan en todo el mundo, pudiendo encontrar productos cuyo funcionamiento está directamente basado en dicho concepto desde los más usuales electrodomésticos (lavadoras, microondas, cámaras fotográficas,...) hasta los más sofisticados sistemas (frenado de trenes, control de hornos, navegación automática,...).

Inicialmente, los conceptos que maneja la Soft Computing eran tratados de forma aislada, indicando el empleo de metodologías fuzzy. La idea de establecer el área de Soft Computing no surgió hasta 1990 (ver en [144]), y fue ya en 1994 [143] cuando Zadeh propuso una primera definición de Soft Computing, estableciéndola en los siguientes términos:

> Básicamente, Soft Computing no es un cuerpo homogéneo de conceptos y técnicas. Más bien es una mezcla de distintos métodos que de una forma u otra cooperan desde sus fundamentos. En este sentido, el principal objetivo de la Soft Computing es apro-

> vechar la tolerancia que conllevan la imprecisión y la incertidumbre, para conseguir manejabilidad, robustez y soluciones de bajo costo. Los principales ingredientes de la Soft Computing son la Lógica Fuzzy, la Neuro-computación y el Razonamiento Probabilístico, incluyendo este último a los Algoritmos Genéticos, las Redes de Creencia, los Sistemas Caóticos y algunas partes de la Teoría de Aprendizaje. En esa asociación de Lógica Fuzzy, Neurocomputación y Razonamiento Probabilístico, la Lógica Fuzzy se ocupa principalmente de la imprecisión y el Razonamiento Aproximado; la Neurocomputación del aprendizaje, y el Razonamiento Probabilístico de la incertidumbre y la propagación de las creencias.

Quedaba así claro que la Soft Computing no estaba definida precisamente, sino que en una primera aproximación se define por extensión, por medio de distintos conceptos y técnicas que intentan superar las dificultades que surgen en los problemas reales que se dan en un mundo que es impreciso, incierto y difícil de categorizar.

Algunos intentos posteriores de ajustar más esta definición de Soft Computing no fueron muy fructíferos. Así, por ejemplo, en [89], a la vista de la dificultad de dar una nueva definición del campo de una manera exacta y consensuada, y de la mayor sencillez de hacerlo por medio de sus características, los autores proponen la siguiente definición de trabajo, que vuelve a ser de tipo descriptivo: "Cualquier proceso de computación que expresamente incluya imprecisión en los cálculos en uno o más niveles, y que permita cambiar (disminuir) la granularidad del problema o suavizar los objetivos de optimización en cualquier etapa, se define como perteneciente al campo de la Soft Computing".

Más recientemente, Verdegay, Yager y Bonissone [134] han presentado una definición más precisa e ilustrativa de lo que es la Soft Computing en la actualidad, en lo siguientes términos:

> El punto de vista que aquí consideramos (y que adoptaremos en el futuro) es otra forma de definir la Soft Computing, por medio de la cual se la considera como la antítesis de lo que podríamos llamar *Hard Computing*. Este punto de vista es consistente con el presentado en [143,144]. La Soft Computing puede, por tanto,

> verse como una serie de técnicas y métodos con las que manejar las situaciones prácticas reales en la misma manera en que los humanos tratan con ellas, es decir, en base a inteligencia, sentido común, consideración de analogías, aproximaciones, etc. En este sentido, Soft Computing es una familia de métodos de resolución de problemas encabezados por el Razonamiento Aproximado y los Métodos de Aproximación Funcional y de Optimización, incluyendo los de búsqueda. Soft Computing está, por tanto, en la base teórica del área de los Sistemas Inteligentes.

Desde mediados de los 90, los Algoritmos Genéticos, o desde un punto de vista general los Algoritmos Evolutivos, se están mostrando como métodos muy valiosos para encontrar buenas soluciones a problemas concretos en estos campos. Como fruto de su atractivo científico, de la diversidad de sus aplicaciones y de la notable eficacia de sus soluciones en el contexto de los Sistemas Inteligentes, se incorporaron a un segundo nivel de las componentes de la Soft Computing.

Los Algoritmos Evolutivos, sin embargo, no son más que una clase más de Heurísticas, o de Metaheurísticas, como también lo son la Búsqueda Tabú, el Enfriamiento (Recocido) Simulado, los métodos de Escalada, la Búsqueda por Entornos Variables (VNS), los Algoritmos de Estimación de Distribuciones (EDA), la Búsqueda Dispersa, los GRASP, la Búsqueda Reactiva, y muchos más. Generalmente, todos estos algoritmos heurísticos (metaheurísticas) suelen proporcionar soluciones que no son las óptimas, pero que satisfacen en buena medida al decisor o usuario. Cuando estos actúan desde el principio de que es mejor satisfacer que optimizar, le dan perfecto sentido en este contexto a la famosa frase de Zadeh: "...en contraste con la computación tradicional (hard), la Soft Computing se beneficia de la tolerancia asociada a la imprecisión, la incertidumbre, y las verdades parciales para conseguir tratabilidad, robustez, soluciones de bajo costo y mejores representaciones de la realidad". Consiguientemente, entre las componentes de la Soft Computing, en lugar de los Algoritmos Evolutivos, que pueden representar sólo una parte de los métodos de búsqueda y optimización que se emplean, deben considerarse también los Algoritmos Heurísticos o aún mejor las Metaheurísticas [134].

El término heurística proviene de la palabra griega "heuriskein", cuyo significado está relacionado con el concepto de encontrar y se vincula a la

famosa y supuesta exclamación ¡eureka! de Arquímedes. Con ese origen se han desarrollado un gran número de procedimientos heurísticos con mucho éxito para la resolución de problemas de optimización, de los que se intenta extraer lo mejor para emplearlos en otros problemas o en contextos más extensos. Esto ha contribuido al desarrollo científico de este campo de investigación y a extender la aplicación de sus resultados. Surgen así las denominadas metaheurísticas, término que apareció por primera vez en un artículo de Fred Glover en 1986 [45].

El término metaheurística deriva de la composición de la palabra heurística con el sufijo meta (más allá o de nivel superior). Aunque no existe una definición formal de qué es una metaheurística, las dos siguientes propuestas dan una representación clara de la noción general del término.

a) Osman y Laporte la definen así [108]: una metaheurística se define formalmente como un proceso iterativo que guía una heurística subordinada, combinando de forma inteligente diferentes conceptos para explorar y explotar el espacio de búsqueda.

b) Según Voss et al. [136]: una metaheurística es un proceso maestro iterativo que guía y modifica las operaciones de heurísticas 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.

Por tanto, parece claro que el concepto de metaheurística tiene un carácter más generalista que el de heurística. Por eso, en lo que sigue, nos concentramos en las primeras, comenzando por puntualizar que, en los términos que las hemos definido, una metaheurística será mejor que otra simplemente en función del rendimiento que proporcionen a la hora de resolver problemas.

Para conseguir el mejor rendimiento de las metaheurísticas, es deseable que tengan una serie de "buenas propiedades", entre las que se encuentran las siguientes [97]:

1. **Simplicidad**, ya que una metaheurística debe estar basada en un principio sencillo y claro que la haga fácil de comprender.

2. **Independencia**, puesto que no puede depender del marco o del agente tecnológico en el que se vaya a desarrollar.

3. **Coherencia**, porque los elementos que caractericen la metaheurística

deben derivarse de forma natural de los principios que la inspiran.

4. **Efectividad**, ya que los algoritmos particulares que se produzcan a partir de las metaheurísticas deben proporcionar soluciones óptimas o muy cercanas a las óptimas. Es decir, de alta calidad en algún sentido propio que cada usuario deberá especificar, y que podrá depender de cada problema concreto.

5. **Eficacia**, en el sentido de fallar sólo en ocasiones muy raras ante casos prácticos del mundo real.

6. **Eficiencia**, como una característica, más que deseable, exigible en términos de recursos, es decir, de tiempo de ejecución, de espacio de memoria y, en definitiva, de costos de desarrollo.

7. **Generalidad**, de forma que se pueda utilizar provechosamente en una gama de situaciones y problemas tan amplia como sea posible.

8. **Adaptabilidad**, para que pueda adecuarse a los diferentes contextos de aplicación y a los distintos casos que se consideran.

9. **Robustez**, ya que no puede ser muy sensible a pequeñas alteraciones del modelo o contexto de aplicación.

10. **Interactividad**, para que el decisor pueda mejorarla a partir de su experiencia y conocimientos.

11. **Diversidad**, para que se permita al usuario elegir entre las distintas soluciones alternativas que proporcione la metaheurística, y

12. **Autonomía**, para facilitar su funcionamiento automático global, o al menos en alguna de las facetas que la caractericen.

A la vista de su definición y de ésta serie de características deseables, es obvio que entre las metaheurísticas, como no podía ser de otra forma, se encuentran los Algoritmos Evolutivos, y que por tanto tienen un fácil acomodo con las demás componentes del segundo nivel de la Soft Computing, con las que sus sinergias deben facilitar la aparición de nuevas metodologías, esquemas y marcos teóricos y prácticos que, como se explica en [144], ayuden a entender y tratar mejor la generalizada imprecisión del mundo real.

Continuando el análisis de las componentes que describen la Soft Computing en los distintos niveles, podríamos concretar que, en el segundo, sus componentes más importantes son el Razonamiento Probabilístico, la Lógica y los Conjuntos Fuzzy, las Redes Neuronales y, a la vista de lo explicado, las Metaheurísticas, que típicamente englobarían los Algoritmos Evolutivos, pero no quedarían circunscritas a estos exclusivamente [134].

Las metodologías que integran la Soft Computing, más que poder ser entendidas de forma aislada, hay que comprenderlas como el resultado de la cooperación, la asociación, la complementariedad o la hibridación de sus componentes de segundo nivel. De la reciente inclusión de las metaheurísticas entre los componentes de la Soft Computing se derivan una serie de facetas teórico prácticas que es importante ver. Las metaheurísticas disponibles son tantas y tan variadas, que es prácticamente imposible ponerse de acuerdo en una forma de clasificarlas que sea universalmente aceptada. No obstante, la jerarquía sobre la que hay más consenso considera tres (cuatro) grupos más destacados:

1. las metaheurísticas para los procedimientos evolutivos, basadas en conjuntos de soluciones que evolucionan en cada generación, siguiendo en muchos casos los mismos principios de la evolución natural,

2. las metaheurísticas para los métodos de relajación, que son métodos de solución de problemas que utilizan adaptaciones del modelo original que son más sencillas de resolver,

3. las metaheurísticas para las búsquedas por entornos, que recorren el espacio de soluciones explotando estructuras de entornos asociadas a esas soluciones, y

4. otros tipos de metaheurísticas que se corresponden con tipos intermedios entre los anteriores, o derivados en algún sentido de estos, con un alto grado de variabilidad.

Si estos cuatro grupos de metaheurísticas los nombramos MH(1), ..., MH(4) respectivamente, podría representarse la Soft Computing y sus principales metodologías con el diagrama descriptivo que se muestra en la figura 1.1, donde, debido a que las componentes clásicas de la Soft Computing (Modelos Probabilísticos, Lógica Fuzzy, Redes Neuronales y Algoritmos

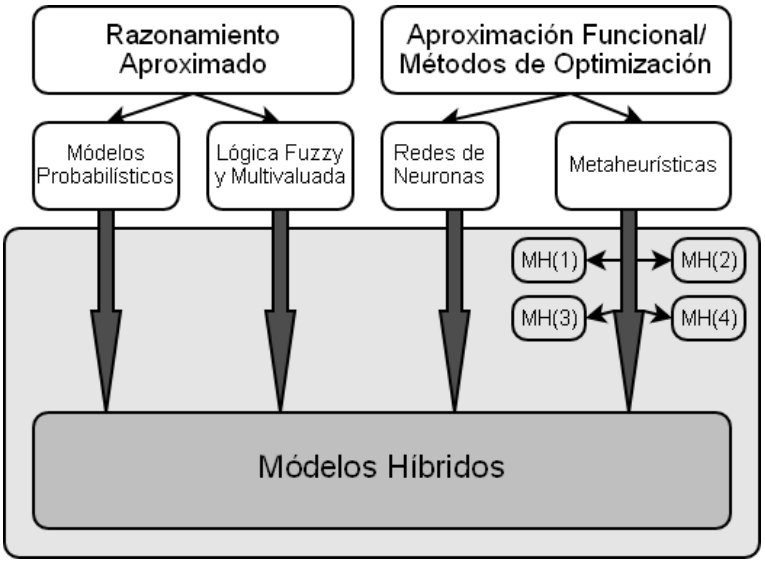


Figura 1.1: Componentes de la Soft Computing.

Evolutivos) siguen estando ahí, las diferentes áreas conocidas y estudiadas se mantienen como hasta ahora, surgiendo como siempre la posibilidad de interrelación de dos o más de las componentes entre sí. Sin embargo, como consecuencia de haber incorporado nuevas posibilidades en la cuarta componente (metaheurísticas) ahora tiene perfecto sentido esperar que aparezcan nuevos Modelos Híbridos que desarrollar.

En las siguientes subsecciones se describen las componentes clásicas de la Soft Computing y la nueva cuarta componente que ahora incluye el campo más genérico de las metaheurísticas donde antes se contemplaban sólo los algoritmos evolutivos.

1.1.1. Modelos probabilísticos

El razonamiento probabilístico (probabilistic reasoning) [117] trabaja con la formación de juicios de probabilidad y creencias subjetivas sobre la base de las probabilidades de que se den ciertos resultados y la frecuencia de ciertos eventos. La base de estas ideas radica en que muchas cosas sólo pueden observarse de manera indirecta y sólo pueden ser predichas de forma parcial. Las decisiones o acciones que se basan en observaciones y predicciones de estos tipos normalmente dependerán de la estimación que hacemos de la

probabilidad de que se den los eventos relevantes, con base a estimaciones como puede ser la de la frecuencia con que se hayan dado en el pasado.

Como en otras áreas de razonamiento y toma de decisiones, hay varias aproximaciones al razonamiento probabilístico: normativa, descriptiva y prescriptiva. Desde el punto de vista normativo, el razonamiento probabilístico está restringido por las mismas reglas matemáticas que gobiernan el concepto clásico de probabilidad de teoría de conjuntos. Se dice que los juicios de probabilidad son coherentes si satisfacen los axiomas de Kolmogorov:

- No hay probabilidades negativas.
- La probabilidad de la tautología es 1.
- La probabilidad de la disyunción de dos sentencias lógicamente excluyentes es igual a la suma de sus probabilidades respectivas.
- La probabilidad de una conjunción de dos sentencias es igual al producto de la probabilidad de la primera y la probabilidad de la segunda asumiendo que la primera es verdadera.

Los primeros tres axiomas anteriores involucran probabilidades no condicionadas mientras que el cuarto introduce probabilidades condicionadas. Cuando se aplica a hipótesis y datos en contextos de inferencia, la simple manipulación aritmética de la regla 4 conduce al resultado de que la probabilidad (a posteriori) de una hipótesis condicionada por datos es igual a la probabilidad del dato condicionado a la hipótesis multiplicada por la probabilidad (a priori) de las hipótesis, dividido todo ello por la probabilidad de los datos. Esta característica es considerada por muchos como un requerimiento normativo del razonamiento probabilístico y su principal diferencia con razonamientos basados en lógica (tanto clásica como multivaluada o difusa).

Entre los modelos para razonamiento probabilístico destacan las redes de Markov y las redes bayesianas que capturan mediante una representación gráfica las propiedades de la estructura de probabilidad de los modelos probabilísticos. Una red de Markov es un grafo no dirigido cuyos enlaces representan dependencias probabilísticas simétricas mientras que las redes bayesianas son grafos dirigidos acíclicos cuyas flechas represesentan dependencias causales o relaciones de tipo clase-propiedad.

En el trabajo con estas redes, cualquier nueva evidencia se ve como una perturbación que se propaga a través de la red, de forma que, al alcanzarse el equilibrio, cada variable estará asociada a un valor fijo, y todas las variables con sus respectivos valores corresponden con la mejor interpretación para la evidencia, dándose el nombre de computación distribuida a esta aproximación [117].

1.1.2. Lógica difusa

La lógica difusa es otra propuesta surgida para la formalización del razonamiento aproximado, que intenta manejar conocimiento propio del "sentido común" (alto, pocos, muchos, caro, etc.). Se trata de una generalización de la lógica booleana (lógica clásica, de verdadero o falso), propuesta, como ya se ha dicho, por Zadeh en 1965 [142]. Consiste en una extensión de la lógica clásica con objeto de permitir manejar el concepto de verdades parciales situadas entre el "completamente verdadero" y el "completamente falso".

En la actualidad [117], la teoría de los conjuntos difusos engloba un corpus bien organizado de nociones básicas incluyendo operaciones de agregación, una teoría generalizada de las relaciones, medidas específicas de cantidad de información y un sistema de cálculo para los números difusos. Tras el concepto de lógica difusa se encuentra la teoría de la posibilidad y los sistemas basados en reglas difusas, que constituyen una herramienta poderosa y versátil tanto para el modelado verbal como el numérico.

En la teoría tradicional de conjuntos, un conjunto A en un universo $\mathcal{U}$ puede representarse por su función característica φ_A que mapea los elementos de $\mathcal{U}$ en el conjunto de dos elementos $\{0, 1\}$. Es decir, para cualquier $x \in \mathcal{U}$ se da que:

$$\varphi_A(x) = \begin{cases} 0 & \text{si } x \notin A \\ 1 & \text{si } x \in A \end{cases} \tag{1.1}$$

En cambio, en el caso de la lógica difusa, un subconjunto difuso A de $\mathcal{U}$ se define mediante una función de pertenencia μ_A que mapea los elementos de $\mathcal{U}$ a un intervalo cerrado $[0, 1]$, de tal forma que, para cada $x \in \mathcal{U}$ se da:

$$\mu_A(x) \in [0, 1] \tag{1.2}$$

En la ecuación anterior (1.2) debe entenderse que, cuanto más cercano a 1 sea el valor de $\mu_A(x)$, mayor es el grado de pertenencia de x a A. Se

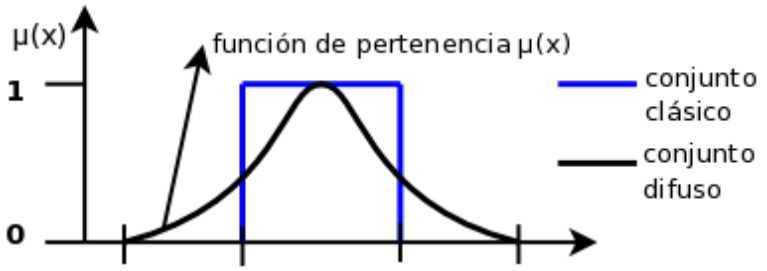


Figura 1.2: Ejemplo de función de pertenencia de un conjunto difuso.

puede ver un ejemplo de función de pertenencia en la figura 1.2, junto a la comparación de lo que podría ser el equivalente en conjuntos clásicos del mismo "concepto". Así, en el caso clásico sólo existen los valores 0 o 1, mientras que en el conjunto difuso el grado de pertenencia variará de forma continua entre 0 y 1. El conjunto representado en la figura podría ser, por ejemplo, el conjunto "estatura media" para la estatura en centímetros en el eje x, de forma que el grado de pertenenencia a "estatura media" comienza a ser mayor que 0 a partir de un cierto número de centímetros y luego, tras alcanzar el valor de 1 a mitad del intervalo, vuelve a decrecer hasta 0 cuando el número de centímetros de estatura alcanza un valor demasiado alto para una estatura media. En el caso de los conjuntos clásicos, el conjunto tiene unos límites más abruptos, pasándose directamente de la no pertenencia a la pertenencia (y viceversa) en valores concretos del eje x, sin ningún grado de pertenencia intermedio antes del cambio. Es habitual este tipo de generalización de los conceptos expresados en términos de conjuntos clásicos transformándolos en conjuntos difusos que toman valor 1 en el centro del rango del conjunto clásico y decrecen luego hasta 0 un poco más allá de sus extremos. De esta forma se suaviza el cambio drástico (y poco acorde a la realidad en muchos casos) entre pertenencia y no pertenencia de los conjuntos clásicos, aunque la forma concreta de la función de pertenencia puede variar mucho, y en lugar de una curva, como en la figura, puede haber funciones triangulares, trapezoidales, etc.

Para procesar los conjuntos difusos se extienden las operaciones de los conjuntos clásicos, de forma que si A y B son conjuntos difusos, el complemento $\bar{A}$, la unión $A \cup B$, y la intersección $A \cap B$ también lo son. Las funciones de pertenencia, en estos casos, se calculan generalmente de la si-

guiente forma:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{1.3}$$

$$\mu_{A \cup B}(x) = max(\mu_A(x), \mu_B(x)) \tag{1.4}$$

$$\mu_{A \cap B}(x) = min(\mu_A(x), \mu_B(x)) \tag{1.5}$$

La lógica difusa se usa ampliamente en modelos de control óptimo, toma de decisiones en condiciones de incertidumbre, para modelar datos de naturaleza vaga o imprecisa y como parte de númerosos métodos de Inteligencia Artificial como puede ser la hibridación con técnicas metaheurísticas.

Recientemente, en la teoría generalizada de la incertidumbre (Generalized Theory of Uncertainty o GTU), presentada en un nuevo artículo de Zadeh [145], se sientan las bases de una teoría para el manejo de incertidumbre que se aleja de las teorías previas que manejan la incertidumbre (la información) de forma puramente estadística. La GTU usa una aproximación más general donde la información se trata como restricciones generalizadas y la incertidumbre estadística es sólo uno de los tipos posibles de incertidumbre. El concepto de restricciones generalizadas parte del hecho de que, en el mundo real, las restricciones no son rígidas, sino que tienen una estructura compleja y son mayormente de naturaleza flexible o elástica. La lógica difusa juega también un papel importante en la GTU, sustituyendo a la lógica bivalente para permitir que todo pueda tener un carácter gradual, o lo que es equivalente, difuso. Sobre estas bases, la GTU se plantea también el objetivo de trabajar con información descrita en lenguaje natural.

1.1.3. Redes neuronales

Una red neuronal [117] es una red que consiste en múltiples unidades de proceso sencillas, cada una de las cuales puede poseer o no una pequeña memoria. Estas unidades se interconectan por ciertos canales de comunicación (conexiones) que generalmente llevan datos que son de tipo numérico (y no simbólico) codificados con alguno de varios métodos posibles. Cada unidad (o neurona) trabaja únicamente con sus datos locales y las entradas que recibe mediante las conexiones, aunque se puede relajar esta restricción durante la fase de entrenamiento para conseguir el correcto funcionamiento de la red.

Aunque en la actualidad no todos los modelos de redes neuronales tienen una inspiración biológica, la mayor parte de la inspiración que llevó a la creación de las redes neuronales vino del deseo de producir sistemas artificiales inteligentes capaces de computaciones sofisticadas (inteligentes, idealmente) similares a las que el cerebro humano realiza de forma rutinaria, pudiendo quizás incluso aumentar nuestra comprensión del cerebro.

Las redes neuronales [93] pueden clasificarse, en cuanto a la topología de la red, de acuerdo con distintos criterios: el número de capas o niveles de neuronas, el número de neuronas por capa y el grado y tipo de conectividad entre las neuronas. En cuanto a las capas, las redes pueden ser monocapa, con una sola capa de neuronas que constituyen a la vez la entrada y la salida de la red, y que presentan interconexiones laterales o recurrentes; o multicapa, con neuronas jerarquizadas en capas, con al menos una capa de entrada y otra de salida y la posibilidad de existir una o varias capas intermedias (ocultas). En las redes multicapa, la conexión entre las neuronas puede ser de dos tipos: con propagación hacia adelante (feedforward), donde las neuronas de una capa reciben las entradas de las neuronas de la capa anterior y mandan sus salidas a la capa siguiente; o con propagación hacia atrás (feedback), donde se permite que las salidas de algunas neuronas se conecten a conexiones de las neuronas de las capas anteriores. La mayor parte de las redes neuronales sufren, además, algún tipo de entrenamiento en el que se ajustan los pesos de las conexiones en base a datos de ejemplo. Se pretende que la red neuronal aprenda de dichos ejemplos qué salida debe dar a las entradas y que posea cierta capacidad de generalización para seguir dando salidas correctas ante datos no conocidos o que no eran parte de los datos de entrenamiento. En la figura 1.3 se puede ver un ejemplo simple de red neuronal multicapa con conexiones feedforwad. Para consultar más ejemplos de redes neuronales y modelos, se recomienda la lectura de [93].

La naturaleza independiente de las neuronas individuales hacen que las redes neuronales sean en alto grado paralelizables y pueden verse, por tanto, como procesadores distribuidos y masivamente paralelos, que están altamente interconectados, que son capaces, “de forma natural”, de almacenar o manejar conocimiento extraído de la experiencia, y que pueden usar dicho conocimiento.

Las redes neuronales son capaces, en principio, de computar cualquier función computable, o de realizar cualquier tipo de función que desempeñe

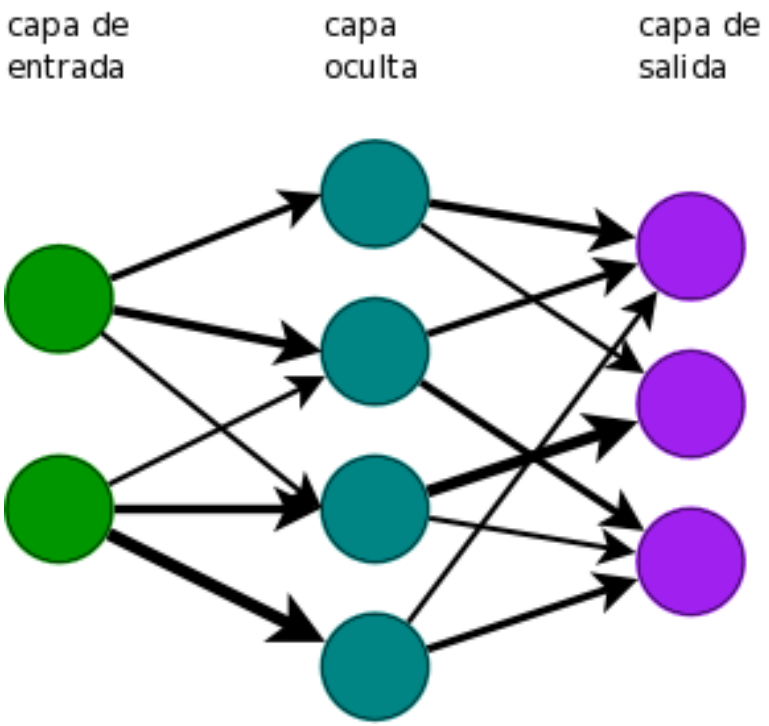


Figura 1.3: Ejemplo de red neuronal multicapa con conexiones feedforward.

un ordenador digital normal, pero en la práctica han mostrado su mayor utilidad en los problemas de clasificación y en los de aproximación de funciones o mapeo, cuando son tolerantes a cierta imprecisión y tienen grandes cantidades de datos de entrenamiento disponibles. No obstante, las técnicas de entrenamiento se basan en la minimización del error cometido, y dicha minimización, pese a que en principio puede acercarse hasta cualquier distancia del mínimo global, en la práctica es computacionalmente intratable si los problemas no son suficientemente pequeños o simples.

1.1.4. Metaheurísticas

Las metaheurísticas, como ya se ha dicho, son una pieza clave en la moderna Soft Computing por la gran variedad de problemas importantes a los que pueden darse soluciones satisfactorias o casi optimales gracias a ellas. Es por ello que, dada su gran importancia, se va a presentar a continuación una breve discusión de las metaheurísticas más usadas y representativas del área [15, 97, 108]. Como vimos en la figura 1.1, las metaheurísticas se ubican en el área de "Aproximación Funcional/Métodos de Optimización" dentro de la estructura general de la Soft Computing, y se vió también una clasificación posible de las metaheurísticas en cuatro grupos (MH(1), MH(2), MH(3) y MH(4)). Daremos, por tanto, en primer lugar, una visión de las metaheurísticas que vamos a presentar posteriormente en términos de estos 4 grupos:

1. MH(1) - Metaheurísticas para los procedimientos evolutivos: De este grupo de metaheurísticas se dará una visión general de **Computación Evolutiva** y se describirá también la **Optimización mediante colonia de hormigas**.

2. MH(2) - Metaheurísticas para los métodos de relajación: No incluimos descripciones detalladas de las metaheurísticas de este tipo por ser en muchos casos implementaciones muy específicas, y no estar entre las metaheurísticas genéricas que mayor relevancia e importancia han tenido.

3. MH(3) - Metaheurísticas para las búsquedas por entornos: De este amplio grupo describiremos la **Búsqueda Local Básica (Mejora Iterativa)**, el **Recocido Simulado (Enfriamiento Simulado)**, la

Algoritmo 1 Mejora iterativa (Búsqueda Local Básica).

```
procedure MejoraIterativa()
  s ← GenerarSoluciónInicial()
  repeat
    s ← Mejorar(N(s))
  until no es posible mejorar
```

Búsqueda Tabú, los **GRASP** y la **Búsqueda por Entornos Variables**.

4. MH(4) - Otros tipos de metaheurísticas que se corresponden con tipos intermedios entre los anteriores o derivados en algún sentido de estos. De este tipo de metaheurísticas no hacemos una descripción completa específica de ninguna, aunque sí que se comentan algunas variantes dentro de las otras metaheurísticas descritas, que pueden considerarse como englobadas en este cuarto grupo.

Los siguientes apartados de esta sección describen en más detalle cada una de las heurísticas anteriores.

Búsqueda Local Básica (Mejora Iterativa)

La Búsqueda Local Básica es una de las primeras metaheurísticas. Se trata de un procedimiento de búsqueda que parte de una solución s y realiza movimientos desde esta solución a soluciones vecinas de la misma según una cierta definición de entorno $N(s)$. Se llama también Mejora Iterativa porque cada nuevo movimiento requiere que la solución resultante sea mejor que la solución actual. El procedimiento para cuando se encuentra una solución mejor que todas sus vecinas, es decir, cuando encuentra un óptimo local. Su esquema general es el presentado en el algoritmo 1.

La función *Mejorar()* escanea el entorno de la solución actual siguiendo desde una estrategia de *primera mejora*, que escoge la primera solución que encuentra en el entorno que sea mejor que la actual, hasta una *mejor mejora*, que recorre el entorno completo para quedarse con la mejor solución, así como cualquier opción intermedia.

La mejora iterativa siempre se detiene al encontrar un óptimo local, con lo cual depende mucho de la elección del entorno y de la solución inicial y no produce, en general, buenas soluciones. Es por ello que una de las ideas

Algoritmo 2 Recocido Simulado.

```
procedure RecocidoSimulado()
  s ← GenerarSoluciónInicial()
  T ← T0
  while no se cumplan las condiciones de parada do
    s' ← EscogerAleatoriamente(N(s))
    if f(s') < f(s) then
      s ← s'
    else
      Aceptar s' como nueva solución con probabilidad p(T,s',s)
    end if
    Actualizar(T)
  end while
```

fundamentales de las metaheurísticas posteriores ha sido el desarrollo de técnicas que permitan evitar quedar atrapados en mínimos locales, ideando mecanismos para escapar de ellos.

En general, las condiciones de parada de una metaheurística no serán tan simples como alcanzar un mínimo local. Entre las posibles condiciones de parada estarán: un tiempo máximo de procesador, un número máximo de iteraciones, encontrar soluciones con un valor objetivo menor que un determinado umbral, un número máximo de iteraciones sin mejorar o criterios más complejos como pueden ser reglas de parada de tipo difuso [11,122,133].

Recocido Simulado (Enfriamiento Simulado)

El Recocido Simulado (o Enfriamiento Simulado) [23, 73] es una de las metaheurísticas más antiguas y sin duda fue uno de los primeros algoritmos que contenía una estrategia explícita para escapar de los mínimos locales. La idea fundamental es permitir movimientos que conducen a soluciones de peor calidad que la solución actual con objeto de escapar del mínimo local. La probabilidad de aceptación de este tipo de movimientos se va decrementando durante la búsqueda, y depende del valor de la solución actual $f(s)$ y de un parámetro de control que modula qué proporción de malas soluciones se aceptan. El esquema general de funcionamiento se describe en el algoritmo 2.

El algoritmo comienza generando una solución inicial (la cual puede ser aleatoria o construida también heurísticamente) e inicializando un parámetro de temperatura (T). En cada iteración se escoge aleatoriamente una

solución s' del entorno $N(s)$ y se aceptará o no como nueva solución actual en función de su valor objetivo ($f(s')$), el valor objetivo de s ($f(s)$) y la temperatura actual (T). De esta manera, s' reemplaza a s siempre que su valor objetivo es mejor o en caso contrario con una probabilidad que va en función de la temperatura y de $f(s') - f(s)$. Esta probabilidad normalmente se calcula haciendo uso de la distribución de Boltzmann ($exp(-\frac{f(s')-f(s)}{T})$) [15].

La temperatura T se va reduciendo a medida que transcurre la búsqueda. Con ello la probabilidad de aceptar soluciones peores es alta al principio y va decreciendo de forma gradual, convergiendo de esta forma a una Mejora Iterativa. Este proceso es análogo al enfriamiento de los metales o el cristal, que los lleva a una configuración de baja energía si son enfriados con una planificación apropiada. La elección de la política de enfriamiento es crucial para el buen rendimiento del algoritmo. Aunque se ha demostrado teóricamente que ciertas políticas de enfriamiento conducen a un óptimo global, el número de iteraciones necesarias para ello es de orden exponencial en el tamaño del espacio de soluciones. Por ello, en aplicaciones prácticas, se emplean políticas más eficientes que no aseguran esta convergencia.

Búsqueda Tabú

La Búsqueda Tabú [46, 47] (Tabu Search o TS) es una metaheurística que consta de 3 fases: búsqueda preliminar, intensificación y diversificación. En la primera de las fases, la Búsqueda Tabú parte de la solución actual s del espacio de búsqueda en que nos encontremos y evalúa la función objetivo sobre todas las soluciones del entorno de s ($N(s)$), encontrando un nuevo punto del espacio de búsqueda s' que es el mejor en $N(s)$, moviéndose a dicha solución incluso si s' es peor que s.

La repetición de esta idea crea la posibilidad de moverse de forma cíclica entre s y s'. Para evitar esto, la Búsqueda Tabú usa la idea de una "lista tabú" de movimientos prohibidos (una lista de movimientos de longitud l en la que el primer elemento en entrar es el primero en salir), que puede verse como una memoria a corto plazo del historial de la búsqueda. De esta forma, una vez que se ha hecho el movimiento $s \rightarrow s'$, el movimiento inverso $s' \rightarrow s$ se prohíbe al menos durante los siguientes l movimientos.

En la segunda fase de la búsqueda (*intensificación*), se comienza con la mejor solución encontrada hasta el momento, limpiando la lista tabú y procediendo como en la búsqueda preliminar por un número determinado de

movimientos. Finalmente, en la fase de *diversificación*, vuelve a limpiarse la lista tabú y se colocan en ella los l movimientos más frecuentemente realizados hasta el momento. A continuación se escoge una solución s aleatoria y se procede como en la búsqueda preliminar por otro número determinado de movimientos. De esta forma, la fase de intensificación actúa como una lupa en las regiones prometedoras descubiertas durante la búsqueda preliminar, y la fase de diversificación fuerza la exploración de regiones totamente nuevas.

La implementación de la lista tabú como una lista que contenga soluciones completas no es práctica, porque el manejo de una lista de dichas características es altamente ineficiente. Por tanto, en lugar de almacenar las soluciones directamente, se almacenan atributos de las mismas. Los atributos pueden ser componentes de las soluciones, movimientos o diferencias entre dos soluciones. Como se puede considerar más de un tipo de atributo, normalmente se tiene una lista tabú para cada uno de ellos. El conjunto de atributos y las correspondientes listas tabú definen las *condiciones tabú* que se usan para filtrar el entorno de una solución y generar el conjunto de soluciones del entorno permitidas (conjunto permitido). Esta estrategia de almacenar atributos en lugar de soluciones es mucho más eficiente, pero significa una pérdida de información, puesto que prohibir un atributo conlleva que probablemente más de una única solución pase a ser tabú, con la posibilidad de que se excluyan del conjunto permitido soluciones de buena calidad no visitadas aún. Para superar este problema se introducen unos *criterios de aspiración*, que permiten incluir una solución en el conjunto permitido incluso si está prohibida por las condiciones tabú. Un criterio usual de aspiración es permitir soluciones que son mejores que la mejor solución alcanzada hasta el momento. En general se permitirán movimientos que serían tabú si a pesar de ello un criterio de aspiración determina que son beneficiosos. Una posible implementación de la búsqueda tabú se muestra en el algoritmo 3, donde k es un contador de la iteración actual que sirve para ilustrar que el conjunto permitido del entorno de una solución s no depende sólo de s sino también de la iteración k en que se esté y los datos que lleva asociados: contenido de las listas tabú, criterios de aspiración actualizados, etc.

Otra forma de extender la Búsqueda Tabú es considerando que las listas tabú no son más que una de las posibles maneras de beneficiarse de la historia de la búsqueda. Otras opciones pasan por el uso de memorias a más largo plazo, tales como memoria de soluciones élite (las mejores soluciones hasta

Algoritmo 3 Búsqueda Tabú.

procedure $TabuSearch()$

s ← GenerarSoluciónInicial()
InicializarListasTabú($TL_1, \ldots, TL_r$)
k ← 0
while no se cumplan las condiciones de parada do
ConjuntoPermitido(s,k) ← {$s' \in N(s)$ | **s' no viola una condición tabú o satisface algún criterio de aspiración**}
s ← ElegirMejor(ConjuntoPermitido(s,k))
ActualizarListaTabúYCriteriosAspiración()
$k \leftarrow k + 1$
end while

el momento completas), novedad de un atributo (la iteración más reciente en que ha entrado en juego), frecuencia (cuántas veces ha sido visitada una solución o atributo), calidad (buenos componentes para soluciones) o influencia (decisiones que se muestran más críticas).

Otra idea para escapar de los mínimos locales y conseguir un buen balance entre exploración y explotación es la propuesta de la Búsqueda Reactiva [10], de la que la Búsqueda Tabú Reactiva (Reactive Tabu Search) es una de las implementaciones usuales. La idea es que la historia de la búsqueda sirva como guía para ajustar los parámetros de la heurística, de forma que se pueda "reaccionar" a la detección de estar en un mínimo local con un reajuste de parámetros que permita escapar de él. En el caso de la RTS este reajuste de parámetros puede corresponderse, por ejemplo, a la variación del tamaño de la lista tabú, ampliándolo cuando se detecte repetición de soluciones y reduciéndolo en caso contrario.

GRASP (Procedimientos de búsqueda voraces, aleatorizados y adaptativos)

Los procedimientos de búsqueda voraces, aleatorizados y adaptativos (GRASP, por sus siglas en inglés: Greedy Randomized Adaptive Search Procedures) [38, 113] son un tipo simple de metaheurísticas que combinan heurísticas de tipo constructivo con búsqueda local. Su estructura general se muestra en el algoritmo 4. GRASP es un procedimiento iterativo que se compone de dos fases: construcción de solución y mejora de solución. Al finalizar el proceso de búsqueda se devuelve la mejor solución encontrada.

Algoritmo 4 GRASP.

procedure $GRASP()$
  while no se cumplan las condiciones de parada do
    s ← **ConstruirSolVorazAleatorizada()**
    AplicarBúsquedaLocal(s)
    MemorizarMejorSoluciónEncontrada()
  end while
procedure $ConstruirSolVorazAleatorizada()$
  s ← $\emptyset$
  α ← **DeterminarLongitudListaCandidatos()**
  while solución no completa do
    LRC_α ← **GenerarListaRestringidaCandidatos(s)**
    x ← **ElegirAleatoriamente**(LRC_α)
    s ← $s \cup \{x\}$
  end while

El mecanismo de construcción de soluciones (función ConstruirSolVorazAleatorizada() del algoritmo 4) se caracteriza por dos ingredientes principales: una heurística constructiva dinámica y la aleatorización. Asumiendo que una solución s no es más que un subconjunto del conjunto de elementos "componentes de solución", la solución se construye paso a paso añadiendo un elemento nuevo cada vez, que se selecciona aleatoriamente de una lista de candidatos. Los elementos a añadir se clasifican de forma ordenada según un criterio heurístico, que les asigna una puntuación en función del beneficio que se obtiene al añadirlos a la solución actual. Dicho beneficio es miope, en el sentido de que no tiene en cuenta el futuro sino sólo el beneficio puntual de añadir cada elemento en el momento actual. La lista de candidatos, llamada *Lista Restringida de Candidatos* (LRC), se construye con los mejores elementos disponibles. Las puntuaciones heurísticas se actualizan en cada paso, de forma que las puntuaciones de los elementos cambian durante la fase de construcción dependiendo de las elecciones posibles. Esta es la razón de que la heurística constructiva se llame dinámica, por contraste con las estáticas, donde la puntuación heurística de los elementos sólo se calcula antes de empezar la construcción. Un ejemplo de heurística dinámica es la heurística de la inserción más barata, donde la puntuación de un elemento se evalúa en función del coste de añadirlo a la solución parcial actual.

La longitud α de la lista restringida de candidatos determina la fuerza de la influencia de la heurística. En el caso extremo en que $\alpha = 1$, se

añadirá siempre el mejor elemento, haciendo que la fase constructiva fuera equivalente a una heurística glotona/voraz (greedy). En el otro caso extremo en que $\alpha = n$, la construcción se vuelve totalmente aleatoria. Por tanto, α es un parámetro crítico que determina el muestreo del espacio de búsqueda, y que puede ser fijado de forma estática o cambiado en cada iteración, tanto de forma aleatoria como por algún esquema adaptativo.

La segunda fase, o fase de mejora de las soluciones, es solamente un proceso de búsqueda local, que puede ser tanto una mejora iterativa como una técnica más avanzada como el Recocido Simulado o la Búsqueda Tabú.

Para que un algoritmo GRASP sea efectivo deben satisfacerse dos condiciones:

- El mecanismo de construcción de soluciones explora las regiones más prometedoras del espacio de búsqueda.
- Las soluciones producidas por la heurística constructiva se encuentran en un valle de soluciones próximas a diferentes soluciones que sean mínimos locales del problema.

La primera condición se puede cumplir eligiendo de forma efectiva la heurística constructiva y una longitud adecuada de la lista restringida de candidatos, mientras que la segunda condición puede alcanzarse eligiendo la heurística constructiva y la búsqueda local de manera que ambas se complementen bien.

Los procedimientos GRASP pueden ser superados muchas veces por otras metaheurísticas más complejas que utilicen, por ejemplo, el historial de la búsqueda. No obstante, su sencillez hace que sean algoritmos generalmente muy rápidos y capaces de producir buenas soluciones en tiempos muy pequeños, lo cual puede ser preferible en muchos contextos, y permite, además, que puedan ser integrados en otras técnicas sin suponer una gran carga para ellas.

Búsqueda por Entornos Variables

La Búsqueda por Entornos Variables (Variable Neighborhood Search o VNS) es una técnica metaheurística que fue presentada por Nenad Mladenovic y Pierre Hansen en [100]. Es una metaheurística relativamente reciente para resolver problemas de optimización cuya idea principal es el cambio sistemático de entorno durante la búsqueda [63, 64]. Es un algoritmo descrito

Algoritmo 5 Búsqueda por Entornos Variables (VNS).

procedure $BusquedaEntornosVariables()$
  Seleccionar estructuras de entorno $N_k, k = 1, ..., k_{max}$
  $s \leftarrow$ **GenerarSoluciónInicial()**
  while no se cumplan las condiciones de parada do
    $k \leftarrow 1$
    while $k < k_{max}$ **do**
      $s' \leftarrow$ **ElegirAleatoriamente**$(N_k(s))$ **// Agitación**
      $s'' \leftarrow$ **BúsquedaLocal**(s')

      if $f(s'') < f(s)$ **then**
        // Movimiento
        $s \leftarrow s''$
        $k \leftarrow 1$
      else
        $k \leftarrow k + 1$
      end if
    end while
  end while

de forma muy general, con lo que deja mucha libertad para el diseño de variantes e implementaciones particularizadas, siendo la estructura presentada originalmente la que se muestra en el algoritmo 5.

El algoritmo VNS comienza con un paso de inicialización en el que se definen un conjunto de estructuras de entorno y se genera una solución. Los entornos pueden escogerse arbitrariamente, pero es frecuente la definición de una secuencia de entornos anidados $N_1 \subseteq N_2 \subseteq \ldots \subseteq N_{k_{max}}$. Tras la inicialización, el algoritmo entra en su bucle principal, que se ejecuta hasta que se cumpla alguna condición de parada y que consiste de 3 fases: *agitación*, *búsqueda local* y *movimiento*. En la fase de agitación se escoge aleatoriamente una solución s' entre las del k-ésimo entorno de la solución actual s. En la fase de búsqueda local se parte de esta solución s' y se realiza una búsqueda local desde ella, la cual puede usar cualquier estructura de entorno, sin estar restringida a las estructuras de entorno $N_k, k = 1, \ldots, k_{max}$. Al final de la búsqueda local, la nueva solución obtenida s'' se compara con s, reemplazando a s si es mejor, y reiniciándose la búsqueda con $k = 1$. En caso contrario, si s'' es peor que s, se incrementa k y se realiza una nueva fase de agitación con un entorno distinto.

El éxito de VNS se basa en tres hechos simples [64]:

1. Un mínimo local con una estructura de entornos no lo es necesariamente con otra.
2. Un mínimo global es mínimo local con todas las posibles estructuras de entornos.
3. Para muchos problemas, los mínimos locales con la misma o distinta estructura de entornos están relativamente cerca.

Esta última observación, que es empírica, implica que los óptimos locales proporcionan información acerca del óptimo global. Puede ser, por ejemplo, que ambas soluciones tengan características comunes. Sin embargo, generalmente no se conoce cuáles son esas características. Es procedente, por tanto, realizar un estudio organizado en las proximidades de este óptimo local, hasta que se encuentre uno mejor.

Los hechos 1 a 3 sugieren el empleo de varias estructura de entornos en las búsquedas locales para abordar un problema de optimización. El cambio de estructura de entornos se puede realizar de forma determinística, estocástica, o determinística y estocástica a la vez.

Volviendo al algoritmo 5, el objetivo de la fase de agitación es perturbar la solución para conseguir un buen punto de partida para la búsqueda local. Dicho punto de partida debería corresponder a la vecindad de un mínimo local diferente del actual, pero no debería estar "demasiado lejos" de s. Esto es así, en primer lugar, porque es probable que, al escoger s' en el entorno de la mejor solución hasta el momento, se consigan soluciones que mantengan algunas buenas características de la actual; en caso contrario, el algoritmo degeneraría en un simple multiarranque aleatorio. Y, en segundo lugar, porque si s' está demasiado lejos de s el algoritmo podría llegar a convertirse en una repetición de búsquedas locales simples con puntos de arranque aleatorios.

El proceso de cambiar entornos en el caso de que no haya mejoras se corresponde con la parte de diversificación del proceso de búsqueda. En particular, la elección de entornos con amplitud creciente conduce a una diversificación progresiva. La efectividad de esta estrategia dinámica de entornos puede explicarse por el hecho de que un mal lugar del espacio de búsqueda con un entorno dado podría ser un buen lugar si se estuviese considerando otro entorno. De la misma forma, una solución que es localmente

óptima en un entorno no tiene por qué serlo en otro. Cada entorno determina unas propiedades topológicas distintas del espacio de búsqueda, definiendo su propio panorama (paisaje). Las propiedades de cada panorama son en general diferentes a las de los demás, con lo que una estrategia de búsqueda se comportará de forma distinta en cada uno de ellos.

Entre las variantes de VNS se encuentran la VNS Descendente (VND), en que la fase de agitación y búsqueda local se sustituyen por una búsqueda local en el entorno N_k, o la VNS general, que mantiene las tres fases del algoritmo 5 con la particularidad de que implementa la fase de búsqueda local usando una VND con entornos N'_j (distintos a los N_k). Se puede hacer otras variantes y extensiones, por ejemplo, permitiendo aceptar soluciones peores con alguna probabilidad (con lo cual se consigue un método ascendente-descendente y no solamente descendente) o realizar movimientos en el mejor entorno $k*$ entre todos los k_{max} entornos (o lo que es lo mismo, escogiendo la mejor mejora).

VNS y sus variantes han sido aplicadas con éxito en problemas basados en grafos, como pueden ser los problema de la p-mediana o del árbol generador mínimo [15].

Computación Evolutiva

Los algoritmos de Computación Evolutiva (Evolutionary Computation o EC) se inspiran en la capacidad de la naturaleza para evolucionar seres vivos bien adaptados a su medio ambiente. Por tanto, de forma sucinta, un algoritmo EC puede ser caracterizado como un modelo computacional de procesos evolutivos. En cada iteración aplican un cierto número de operadores a los individuos de la población actual para generar la población de la siguiente generación (iteración). Generalmente, los algoritmos EC usan operadores llamados *recombinación* o *cruce (crossover)* para combinar dos o más individuos y producir otros individuos nuevos; así como operadores de *mutación* o *modificación* que contribuyen tanto a aumentar la diversidad como a que se produzca una autoadaptación de los individuos con mejores "genes". La fuerza conductora en los algoritmos evolutivos es la selección de individuos basados en su adaptación (fitness, que puede ser el valor de una función objetiva, el resultado de un experimento de simulación u otra clase de medida de calidad). Los individuos con una mejor adaptación tienen una probabilidad más alta de ser escogidos como miembros de la población en

Algoritmo 6 Computación Evolutiva.

procedure *ComputacionEvolutiva*()
  $P \leftarrow$ **GenerarPoblaciónInicial()**
  Evaluar(P)
  while no se cumplan las condiciones de parada do
    $P' \leftarrow$ **Recombinar**(P)
    $P'' \leftarrow$ **Mutar**(P')
    Evaluar(P'')
    $P \leftarrow$ **Seleccionar**($P'' \cup P$)
  end while

la siguiente iteración (o como padres de la nueva generación de individuos). Esto se corresponde con el principio de la supervivencia de los más adaptados en la evolución natural. Es precisamente la capacidad de lo natural de adaptarse a un entorno cambiante lo que sirvió de inspiración para los algoritmos EC.

Ha habido una variedad de algoritmos EC ligeramente distintos propuesta a lo largo de los años, entre las que cabe destacar la programación evolutiva (Evolutionary Programming o EP) [40,41], las estrategias evolutivas (Evolutionary Strategies o ES) [118] y los algoritmos genéticos (Genetic Algorithms o GA) [48,67,99,119,135]. Aunque EP se propuso originalmente para operar en representaciones discretas de máquinas de estados finitos, la mayoría de las variantes actuales se usan para optimización continua, como también es el caso actual de ES, mientras que los GA se aplican principalmente para resolver problemas de optimización combinatoria.

La estructura general de un método de computación evolutiva se puede ver en el algoritmo 6, donde P representa la población de individuos. Tras generar y evaluar la población inicial, en cada iteración se genera una nueva generación P' de descendientes de P mediante recombinación (P') y mutación (P''). Finalmente, en cada iteración se selecciona una nueva población P entre los individuos de la antigua P y los descendientes en P''.

Los componentes principales de un algoritmo EC son [15]:

- *Individuos de la población*: Los algoritmos EC manejan poblaciones de individuos. Generalmente, estos individuos se corrresponden con soluciones del problema que se resuelve, pero también pueden ser soluciones parciales, conjuntos de soluciones o cualquier tipo de objeto

que pueda convertirse de manera estructurada en una o más soluciones. En optimización combinatoria son comunes las representaciones de las soluciones como una cadena de bits (lo que puede verse como genes que pueden estar activos "1" o inactivos "0") y las permutaciones de n enteros.

- *Proceso Evolutivo*: En cada iteración tiene que decidirse, mediante algún esquema de selección, qué individuos forman parte de la población de la siguiente iteración. Este esquema de selección puede ir desde escoger solamente los descendientes de la generación anterior (reemplazo generacional) hasta la inclusión de los individuos de la población anterior y de los descendientes en distintas proporciones. En el caso de poblaciones de tamaño fijo es habitual mantener al menos los mejores individuos en la nueva población. Si la población es de tamaño variable, una de las condiciones de parada puede ser que sólo quede un elemento en la población.

- *Estructura de entorno*: Se utiliza una función de entorno que asigna a cada individuo un conjunto de individuos con los que le está permitido combinarse para producir descendientes. Si se permite la combinación de un individuo con cualquier otro, se habla de poblaciones no estructuradas, siendo poblaciones estructuradas en caso contrario. Un ejemplo de algoritmo EC con poblaciones estructuradas es el algoritmo genético paralelo propuesto por Mühlenbein [103].

- *Fuentes para la recombinación*: La forma más común de crear descendientes mediante la recombinación es utilizando una pareja de padres (cruce de dos padres), pero también hay operadores de recombinación que utilizan más de dos padres (cruce multipadre) [34]. Algunos desarrollos usan estadísticas poblacionales para generar los individuos de la siguiente generación [104, 128].

- *Infactibilidad*: Dado que la recombinación de individuos puede potencialmente producir individuos que no sean factibles según las restricciones del problema, un aspecto importante de un algoritmo EC es cómo maneja a los individuos no factibles. Hay tres formas principales de tratar estas situaciones, siendo la más sencilla rechazar a dichos individuos no factibles. Pero en algunos problemas (como la asignación

de horarios) puede ser muy difícil encontrar individuos factibles, con lo que otra estrategia posible es mantener a dichos individuos pero penalizarlos en la función que mide su calidad. La tercera estrategia es la de tratar de reparar los elementos no factibles, utilizada por ejemplo por Eiben [35].

- *Estrategia de intensificación*: Se ha observado que, en muchas aplicaciones, es beneficioso introducir mecanismos que mejoren la adaptación (fitness) de los individuos. Los algoritmos EC que aplican una búsqueda local se llaman normalmente algoritmos meméticos [66, 101, 102]. La búsqueda local ayuda a identificar rápidamente buenas áreas del espacio de búsqueda. Otra estrategia de intensificación es usar operadores de recombinación que explícitamente traten de combinar partes "buenas" de los individuos (en lugar de un simple punto de cruce para cadenas de bits), de forma que la búsqueda se conduzca hacia individuos con "buenas" propiedades. Esta estrategia se conoce como aprendizaje de unión (linkage lerning) o aprendizaje de bloques de construcción (building block learning) [65, 137].

- *Estrategia de diversificación*: Una de las principales deficiencias de los algoritmos EC (especialmente cuando se usa búsqueda local) es la convergencia prematura a soluciones subóptimas. El mecanismo más simple para diversificar el proceso de búsqueda es el uso de un operador de mutación, que, en su forma más sencilla, simplemente realiza una pequeña perturbación aleatoria en el individuo. Entre las primeras estrategias usadas para conseguir diversificación se encuentran también la aglomeración (crowding) o su pariente cercano la preselección (ver en [92]); y también se ha conseguido la diversificación mediante la reduccción de la puntuación para reproducción de un individuo en función de lo densamente poblada que esté su zona del espacio de búsqueda [49].

Optimización mediante colonia de hormigas

La optimización mediante colonia de hormigas, conocida también como ACO (de sus siglas en inglés: Ant Colony Optimization), es un enfoque metaheurístico propuesto por Dorigo y otros [26–28, 31].

Las colonias de hormigas y, de forma más general, las comunidades de

insectos sociales, son sistemas distribuidos que, a pesar de la simplicidad de sus individuos, presentan una organización social muy estructurada. Como resultado de esta organización, las colonias de hormigas pueden completar tareas complejas que en algunos casos exceden notablemente las capacidades de una única hormiga. Los algoritmos basados en hormigas estudian modelos derivados de la observación del comportamiento de hormigas reales, y usan estos modelos como inspiración para el diseño de nuevos algoritmos para la solución de problemas de optimización y de control distribuido.

La idea principal es que los principios de autoorganización que permiten el comportamiento altamente coordinado de las hormigas reales pueden ser explotados para coordinar poblaciones de agentes artificiales que colaboran para resolver problemas computacionales. Varios aspectos del comportamiento de las hormigas han inspirado diferentes tipos de algoritmos basados en hormigas. Entre estos aspectos cabe mencionar la búsqueda de comida, la división de trabajo, clasificación de crías y transporte cooperativo. En todas estas actividades las hormigas coordinan sus actividades mediante estigmergia, un concepto introducido por el zoólogo francés Pierre-Paul Grassé en 1959 [61]. Grassé la definió como "Estimulación de trabajadores por el rendimiento que han alcanzado". La palabra deriva del griego *stigma* (mancha o signo) y *ergon* (acción) y captura la noción de que las acciones de un agente dejan signos en el entorno que otros agentes perciben y que pueden determinar sus acciones posteriores. Los biólogos han mostrado que muchos comportamientos a nivel de colonia que se observan en insectos sociales pueden ser explicados por modelos muy simples en los que sólo la comunicación estigmérgica está presente. En otras palabras, se ha mostrado que a menudo basta con considerar comunicación indirecta por estigmergia para explicar cómo los insectos sociales pueden conseguir su autoorganización. Por ejemplo, en muchas especies de hormigas, las que caminan hacia o desde una fuente de comida depositan en la tierra una sustancia química llamada feromona. Otras hormigas perciben la presencia de feromona y tienden a seguir los caminos con la mayor concentración de feromona. De esta forma, las hormigas son capaces de transportar comida a su nido de forma remarcablemente efectiva, aproximándose a seguir el camino más corto y adaptándose a la aparición de obstáculos. La idea que subyace en los algoritmos basados en colonias de hormigas es usar una forma de *estigmergia artificial* para coordinar una colonia de agentes artificiales.

Algoritmo 7 Colonia de hormigas (ACO).

procedure *ColoniaHormigas*()
  Fijar parámetros e inicializar las marcas de feromona
  while no se cumplan las condiciones de parada do
    ConstruirSolucionesDeHormigas()
    AplicarBúsquedaLocal() (*opcional*)
    ActualizarFeromona()
  end while

Los algoritmos de colonias de hormigas se basan en los modelos desarrollados para explicar el comportamiento de hormigas reales enfrentadas a la alternativa de usar dos puentes distintos (de igual o diferente tamaño) para llegar a una fuente de comida [25, 60, 109]. Goss [60] da una formulación matemática de dichos modelos. Asumiendo que, en un tiempo dado, m_1 hormigas habrán usado el primer puente y m_2 hormigas habrán usado el segundo, la probabilidad p_1 de que una hormiga tome el primer puente vendrá dada por la ecuación:

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h} , \tag{1.6}$$

donde los parámetros k y h deben ser fijados de acuerdo a los datos experimentales. Por ejemplo, se han obtenido muy buenos ajustes en simulaciones de tipo Monte Carlo para $k \approx 20$ y $h \approx 2$ [109]. Por otro lado, como es lógico, la probabilidad de que las hormigas tomen el segundo puente sería simplemente $p_2 = 1 - p_1$.

ACO es una metaheurística de tipo constructivo, en la que la solución se forma probabilísticamente al ir añadiendo componentes de soluciones parciales considerando las trazas de feromona y elecciones heurísticas para el problema particular. El modo general de funcionamiento de los algoritmos ACO se muestra en el algoritmo 7. Tras la inicialización, ACO itera sobre tres fases: en cada iteración, las hormigas construyen un determinado número de soluciones, las cuales son mejoradas mediante búsqueda local (esta fase es opcional) y finalmente se actualizan las trazas de feromonas. Más detalladamente estas fases son:

- *ConstruirSolucionesDeHormigas*: Un conjunto de m hormigas artificiales construye iterativamente soluciones añadiendo elementos de un

conjunto finito de componentes de solución C. La construcción comienza con una solución parcial vacía $s = \emptyset$. En cada paso de la construcción, la solución parcial s se extiende mediante la adición de un componente de solución factible del conjunto $N(s) \subseteq C$, que se define como el conjunto de componentes que pueden añadirse a la solución parcial actual s sin violar ninguna restricción. De esta forma, el proceso de construcción de soluciones puede verse como un camino en el grafo de construcción $G_C(V, E)$. Este grafo puede obtenerse del conjunto C de componentes de solución de dos maneras: representando las componentes por vértices o por aristas. La elección de una componente de solución de $N(s)$ se guía por un mecanismo estocástico, que está influenciado por la feromona asociada a cada uno de los elementos de $N(s)$. La regla para la elección estocástica de componentes de solución varía entre diferentes algoritmos ACO, pero en todos ellos se inspira en el modelo del comportamiento de las hormigas reales dado por la ecuación 1.6.

- *AplicarBúsquedaLocal*: Una vez que se han construido las soluciones, y antes de actualizar la feromona, es común mejorar las soluciones obtenidas por las hormigas realizando una búsqueda local. Esta fase es muy dependiente del problema específico, y es opcional, aunque se incluye en muchos de los últimos algoritmos ACO.

- *ActualizarFeromona*: El objetivo de la actualización de feromona es incrementar los valores de feromona asociados a las componentes de las soluciones buenas o prometedoras y decrementar aquellos asociados a las componentes de las soluciones malas. Normalmente esto se consigue decrementando todos los valores ("evaporación de feromona") e incrementando los niveles de feromona asociados con un conjunto escogido de buenas soluciones.

Entre los principales algoritmos basados en hormigas, se encuentra el Sistema Hormiga (Ant System o AS) [26, 30] que fue el primer algoritmo de este tipo que se desarrolló y cuya característica principal es que todas las hormigas intervienen en la actualización de los valores de feromona. Las dos variantes principales de este algoritmo original son el Sistema Hormiga $MAX - MIN$ ($MAX - MIN$ Ant System o MMAS) y el Sistema Colonia de Hormigas (Ant Colony System o ACS). MMAS [127] es una mejora sobre

el algoritmo AS original. Sus elementos característicos son que sólo la mejor hormiga actualiza las trazas de feromona y que el valor de feromona está acotado, con valores de cotas determinados de forma empírica y adaptados a cada problema particular. ACS [29] presenta como contribución principal la introducción de una actualización local de feromona de forma adicional a la actualización de feromona realizada al final del proceso de construcción (llamada actualización de feromona fuera de línea). La actualización local de feromona se realiza por todas las hormigas después de cada paso de la construcción y cada hormiga la aplica únicamente a la última arista recorrida. El objetivo principal de esta actualización local es diversificar la búsqueda realizada por las siguientes hormigas durante una iteración, decrementando la cantidad de feromona en las aristas recorridas, de forma que se estimula a las siguientes hormigas a escoger otras aristas que produzcan soluciones diferentes, reduciendo la posibilidad de aparición de soluciones idénticas en una iteración dada.

1.2. Sistemas de Ayuda a la Decisión

Muchos problemas a los que debemos enfrentarnos son extremadamente complejos por la presencia de varias fuentes de incertidumbre, varios objetivos y metas conflictivos, posibles impactos de las decisiones a largo plazo,... Aunque en ocasiones es posible resolverlos mediante la experiencia y la intuición, o de forma manual, se ve repetidamente probado que aproximarse así a los problemas complejos puede conducir a malas soluciones [121].

La disponibilidad de diversas formas de solucionar problemas con una alta calidad, aparte del valor intrínseco que conlleva, supone una ventaja cuando dichas soluciones están disponibles para los expertos o decisores que tienen que abordar cada problema, permitiéndoles disponer de un mayor nivel de conocimiento e información que los guíe en la toma acertada de decisiones. Pero, una vez más, la realización totalmente manual de estas tareas es un trabajo tedioso y complicado, especialmente cuando estamos abordando problemas en los que el volumen de datos disponibles es muy grande, como es el caso de los problemas de Bioinformática.

Los Sistemas de Ayuda a la Decisión (SAD) [123], surgidos a comienzos de la década de los 70, son soluciones informáticas que pueden usarse para ayudar en la toma compleja de decisiones y la resolución de problemas de

forma estructurada. Sobre este planteamiento, el diseño clásico de SAD se compone de componentes para: (i) manejo de datos tanto internos como externos, información y conocimiento; (ii) funciones potentes de modelado y (iii) el diseño de interfaces de usuario sencillas pero poderosas. En general, la investigación desarrollada en este área se ha centrado generalmente en cómo se puede mejorar la eficiencia y la efectividad con la que un usuario toma una decisión haciendo uso de tecnologías de la información.

Desde el origen de los SAD, las tecnologías de la información han evolucionado mucho y, consecuentemente, los SAD también han evolucionado sustancialmente, incorporando en su diseño muchas técnicas y elementos nuevos, como los almacenes de datos (data warehouses), el procesamiento analítico en línea (On-Line Analytical Processing u OLAP), la minería de datos (data mining) y las tecnologías web. Los SAD son un campo tecnológicamente rico cuya naturaleza e historia se percibe desde distintos puntos de vista por diferentes personas, con nociones diferentes de qué ha pasado hasta ahora en el campo y qué es importante (ver [4, 96, 114, 115] para más información).

Un SAD puede componerse generalmente de varios subsistemas [132]:

1. *Subsistema de manejo de datos*: Contiene una base de datos con los datos relevantes para la situación, que es manejada por un **sistema gestor de base de datos (SGBD)**.

2. *Subsistema de manejo de modelos*: Incluye modelos que proporcionan las capacidades analíticas del sistema, así como el software para su manejo, que suele llamarse **sistema gestor de la base de modelos (SGBM)**. Los modelos pueden estar definidos mediante un lenguaje de modelado o ser simplemente métodos o algoritmos de resolución del problema.

3. *Subsistema de manejo de conocimiento*: Este subsistema está presente en los SAD más avanzados, y puede ser tanto un sistema de apoyo a los otros subsistemas como un componente independiente. Contiene un **sistema gestor de la base de conocimiento (SGBC)**, y proporciona funciones inteligentes que complementan la inteligencia propia del decisor. Puede tener interconexión con una base de conocimientos a nivel de empresa.

4. *Subsistema de interfaz de usuario*: Es el medio mediante el cual el usuario puede comunicarse y controlar el SAD. El usuario se considera parte del propio sistema, y muchas de las contribuciones únicas hechas en el terreno de los SAD derivan de la interacción intensiva entre el ordenador y el decisor.

Existen muchas maneras de clasificar los SAD, y el tipo de SAD condiciona en buena medida su diseño y estructura. Entre los muchos tipos posibles de SAD, y aunque no vamos a describirlos detalladamente todos, se incluyen, a continuación, y a modo ilustrativo, los siguientes [132]:

- SAD Orientado a Textos (Text-Oriented DSS): Los datos y el conocimiento y cualquier otro tipo de información suelen encontrarse en formato texto. Un SAD orientado a textos ayuda al decisor manteniendo electrónicamente representaciones textuales de la información que puede influir en sus decisiones. Los textos pueden ser creados, modificados y recuperados cuando sea necesario, y las aplicaciones para SAD de este tipo pueden incorporar agentes inteligentes y tecnologías web como hipertexto y documentos con imágenes.

- SAD Orientado a Bases de Datos (Database-Oriented DSS): Los primeros SAD usaban sistemas de base de datos sencillos o distintas configuraciones de bases de datos relacionales. Un SAD orientado a bases de datos contempla, además, capacidades potentes de consulta y de generación de informes.

- SAD Orientado a resolutores (Solver-Oriented DSS): Un resolutor es un algoritmo o procedimiento desarrollado como un programa informático para resolver un tipo particular de problema. Un tipo particularmente simple de estos SAD son los SAD Orientados a Hojas de Cálculo (Spreadsheet-Oriented DSS) que utilizan resolutores (llamados generalmente funciones en este caso) como los incluidos en programas como Microsoft Excel, que junto a sus capacidades propias de base de datos o la posibilidad de conexión a base de datos externas, puede usarse como base para construir este tipo de SAD. Por otro lado, los SAD basados en Optimización (Optimization-based DSS) pueden verse también como un tipo de estos SAD o como un tipo más general con tres etapas: *Formulación*,

Solución (que es donde irían los resolutores) y *Análisis*. Estos SAD se verán en más detalle posteriormente ya que son los que consideramos más adecuados para los objetivos de esta tesis.

- SAD Orientado a Reglas (Rule-Oriented DSS): Un SAD orientado a reglas incluye reglas tanto de procedimiento como de inferencia (razonamiento), normalmente en forma de un Sistema Experto (Expert System), que no es más que un sistema que emula el comportamiento de los expertos de un dominio concreto, para conseguir una mejor calidad y rapidez en las respuestas, aumentando la productividad.

- SAD Compuestos (Compound DSS): Un SAD compuesto es un SAD híbrido que incluye dos o más de los tipos de SAD anteriores.

- SAD Inteligentes (incluye, por ejemplo, SAD que se adaptan al usuario, SAD Grupales (para dar soporte a decisiones en grupo), SAD Institucionales (que se aplican en tomas de decisiones de naturaleza recurrente),...

Entre todas las clases de SAD, esta tesis se concentra, pues, en un área particularmente interesante e importante: los Sistemas de Ayuda a la Decisión basados en Optimización (Optimization-based Decision Support Systems). Como se puede leer en [123], los SAD basados en optimización pueden dividirse en tres etapas: *Formulación*, *Solución* y *Análisis*. La etapa de *Formulación* trata con la generación de un modelo en una forma aceptable por un resolutor del modelo (solver); la etapa de *Solución* se asocia a la resolución algorítimica del modelo por un resolutor apropiado; y la etapa de *Análisis* engloba el análisis e interpretación de una solución o un conjunto de soluciones.

Cuando tratamos con problemas diferentes o incluso, a veces, con un único problema, pueden existir diversos modelos distintos y cada investigador / desarrollador puede decidir escribir sus propios algoritmos (resolutores) en cualquier lenguaje de programación que escojan, con parámetros y entradas en formatos posiblemente distintos. Este es el caso, por ejemplo, en muchos problemas de Bioinformática, como son los problemas de comparación de proteínas abordados en esta tesis. Debido a estas razones, un resolutor dado puede no ser capaz de encajar en un determinado SAD basado en optimización, incluso aunque dicho SAD sea para el mismo problema sobre el que

el resolutor está diseñado para trabajar. Por tanto, el decisor (investigador) está forzado a ignorar dichos resolutores "no compatibles" o manejarlos de forma independiente al SAD, con las contrariedades que ello conlleva.

Pese a que se ha dedicado cierto esfuerzo en la integración modelo-resolutor y la independencia de datos [82, 83, 91], hay varias limitaciones importantes incluso en las propuestas más recientes [83]. Entre ellas se encuentran tener que registrar y manejar manualmente reglas de interfaces o tener que resolver conflictos estructurales entre los modelos y los resolutores mediante la escritura de scripts que tranformen los datos. Estas limitaciones dificultan el mantenimiento y la utilización de estos sistemas por no expertos y también implican la inversión de una cantidad no despreciable de tiempo y esfuerzo cuando se necesita añadir un nuevo modelo o resolutor.

Considerando estos inconvenientes, sería deseable para un SAD de este tipo tener las siguientes características y funcionalidades:

- Facilitar la incorporación de resolutores preexistentes sin tener que reimplementarlos en ningún lenguaje específico, lo cual es necesario porque los usuarios del SAD no estarán en general dispuestos a realizar esta tarea o podrían carecer del conocimiento y del tiempo necesarios para ello.

- Recolectar y mostrar al usuario cualquier resultado obtenido por los resolutores de manera sencilla y sistemática.

- Permitir agregar, modificar y borrar resolutores de un modo dinámico, incluyendo la generación dinámica de las interfaces gráficas de usuario (GUI) necesarias para la ejecución de los resolutores y el análisis de sus resultados.

- No requerir la recompilación de los resolutores o del SAD durante el trabajo normal con el sistema, incluyendo las tareas relacionadas con las características previas.

- Proporcionar una base de datos de resolutores que permita reutilizarlos o modificarlos en cualquier momento, para poder olvidarse de los detalles específicos de configuración y parámetros de los resolutores y concentrarse en su uso y el análisis de sus resultados.

- Debería estar basado en herramientas de código abierto y lenguajes

bien establecidos, de forma que el SAD en sí pueda ser fácilmente extendido o portado a distintas plataformas.

- Debería poder ejecutarse en los principales ordenadores y sistemas operativos, de forma que no se imponga una restricción al entorno de trabajo de un decisor. Así se esquivaría el inconveniente de tener resolutores que sólo funcionan en una determinada plataforma, al poder llevar el SAD a ella.

Es por ello que, como parte de los objetivos de esta tesis, se ha pretendido desarrollar y poner a disposición de los investigadores un SAD basado en optimización, genérico y extensible de forma dinámica, con la capacidad para proporcionar las características y funcionalidades deseables detalladas en la lista anterior. Esta herramienta es esencialmente un esqueleto (framework) para desarrollar SAD basados en optimización, llamado SiGMA, cuya descripción, características, y el modo de usarlo de base para un SAD en un problema específico, se presentan en el capítulo 3.

1.3. Resumen

En este capítulo se ha presentado el área de Soft Computing, desde sus componentes clásicas hasta la visión más moderna en que las metaheurísticas en general pasan a formar una parte importante de sus componentes. Así se abre el campo de la investigación en este área al estudio de un abanico mucho más amplio en la parte de métodos de optimización y aproximación funcional de los que incluían las definiciones previas de Soft Computing, que limitaban a los Algoritmos Evolutivos los métodos heurísticos aplicados.

Como vimos en la introducción, esta tesis hará uso de varias técnicas del área de Soft Computing para aplicarlas en la comparación de estructuras de proteínas, y la introducción del área que se ha realizado en este capítulo sirve para enmarcar el trabajo posterior de los capítulos 4 y 5 en donde se presentan los métodos y estrategias concretos desarrollados. En particular, entre todas las técnicas que se engloban en el área de Soft Computing, se hará un uso especialmente significativo de los conjuntos difusos y de las metaheurísticas, a través de la implementación de un método basado en Búsqueda por Entornos Variables.

Así mismo, se ha realizado una pequeña introducción a los Sistemas de Ayuda a la Decisión (SAD), que han sido de vital importancia en el manejo

de los métodos de comparación utilizados a lo largo de toda la experimentación realizada durante la tesis, todo ello gracias al SAD que será descrito en el capítulo 3.

Capítulo 2

Bioinformática

El Proyecto Genoma Humano (Human Genome Project) tenía como objetivo determinar las posiciones relativas de todos los pares de bases (nucleótidos) que componen el genoma humano e identificar los entre 20000 y 25000 genes presentes en él. Era un objetivo ambicioso, iniciado en 1990 por el Departamento de Energía y los Institutos de la Salud de los Estados Unidos, con un gran presupuesto de 3000 millones de dólares y un plazo de realización de 15 años. El proyecto fue un éxito y, gracias a los avances tecnológicos, se pudo completar en 2003, dos años antes de lo previsto.

Los datos de la secuencia completa del genoma humano obtenidos en el Proyecto Genoma están disponibles públicamente. No obstante, el conocimiento del genotipo completo de un organismo es únicamente un primer paso para la comprensión del mismo y de su fenotipo, y la finalización del Proyecto Genoma, lejos de suponer el final de la Genómica, abrió una vía para obtener beneficios en los campos de la medicina y la biotecnología, constituyendo los datos del Proyecto Genoma una poderosa herramienta para esta investigación.

El beneficio económico potencial de los avances en el descubrimiento de nuevos fármacos, por ejemplo, hace que las inversiones en estos campos sean muy altas, y la disponibilidad de dinero da lugar a la realización de numerosas investigaciones.

La explosión de datos que ha supuesto el Proyecto Genoma y todas las nuevas investigaciones han dado origen a la disciplina de la Bioinformática, que intenta servir como herramienta para el manejo de estos datos y la realización de experimentos. No obstante, la definición concreta de Bioinformática es una materia de cierto debate, desde los que la definen de manera

restrictiva como el desarrollo de bases de datos para almacenar y manipular información genómica, hasta los que la definen de manera amplia abarcando toda la biología computacional. Si nos centramos en el uso reciente en la literatura [16], la Bioinformática se ocupa de dos flujos de información de la biología molecular:

- El primer flujo de información está basado en el dogma central de la biología molecular: las secuencias de ADN se transcriben en secuencias de mARN, que se traducen en estructuras de proteínas, las cuales se pliegan en estructuras tridimensionales (3D) que realizan determinadas funciones. Estas funciones son seleccionadas, en un sentido Darwiniano, por el entorno del organismo, que impulsa la evolución de las secuencias de ADN dentro de una población. La primera clase de aplicaciones Bioinformáticas se enfoca en la transferencia de información entre cualquiera de las fases de este flujo, incluyendo la organización y control de genes en la secuencia de ADN, la identificación de unidades de transcripción en el ADN, la predicción de estructuras de proteínas a partir de la secuencia y el análisis de la función molecular.
- El segundo flujo de información se basa en el método científico: creamos hipótesis referidas a la actividad biológica, diseñamos experimentos para probar estas hipótesis, evaluamos los datos resultantes por compatibilidad con las hipótesis, y extendemos o modificamos las hipótesis en respuesta a los datos. La segunda clase de aplicaciones Bioinformáticas cubre la transferencia de información dentro de este protocolo, incluyendo sistemas que generan hipótesis, diseñan experimentos, almacenan y organizan los datos de los experimentos en bases de datos, testean la compatibilidad de los datos con los modelos, y modifican las hipótesis.

De forma alternativa, podemos decir que la Bioinformática se encarga del desarrollo y aplicación de algoritmos y métodos para transformar datos en conocimiento del sistema biológico. Para ello han de proporcionarse herramientas que permitan cruzar información de secuencias, estructuras, microarrays, información textual, etc., de manera que la integración de esta información pueda conducir a la creación de conocimiento nuevo.

El interés en Bioinformática ha estado motivado por la aparición de técnicas experimentales que generan datos con un alto nivel de rendimien-

to, tales como la secuenciación del ADN, la espectroscopía de masas, o el análisis de expresión en microarrays [1, 80, 98]. Los problemas en los que se aplica la Bioinformática son variados: diseño de bases de datos, comparación y alineamiento de secuencias y de estructuras, construcción de árboles filogenéticos, ensamblado de fragmentos, mapeo físico de cromosomas, rearreglo de genomas, predicción de estructuras proteicas, análisis de perfiles de expresión génica en microarrays, etc. El rápido incremento en el número de estructuras macromoleculares 3D, disponibles en bases de datos como el Protein Data Bank (PDB) [14], ha conducido a la aparición de una subdisciplina de la Bioinformática: la Bioinformática estructural, que se ocupa de la representación, almacenamiento, recuperación, análisis y visualización de información estructural a escala atómica y subcelular.

La Bioinformática estructural, como otras muchas subdisciplinas de la Bioinformática, se caracteriza por dos metas u objetivos: la creación de métodos de propósito general para la manipulación de información sobre macromoléculas biológicas, y la aplicación de estos métodos a la resolución de problemas en Biología y la creación de conocimiento nuevo. Estas dos metas están profundamente interconectadas, porque parte de la validación de los nuevos métodos consiste en comprobar su éxito al resolver problemas reales. Al mismo tiempo, los desafíos actuales en Biología demandan el desarrollo de nuevos métodos que puedan manejar el volumen de datos actualmente disponible y la complejidad de los modelos que los científicos tienen que crear para explicar estos datos.

La creciente disponibilidad de datos de secuencias ha sido un imán para los interesados en el análisis de cadenas, los algoritmos y los modelos probabilísticos [32, 62], donde los mayores logros han sido el desarrollo de algoritmos para alineamiento de pares de secuencias y para alineamiento múltiple, la definición y descubrimiento de motivos de secuencias y el uso de modelos probabilísticos como los modelos ocultos de Markov para encontrar genes [18], alinear secuencias [70] y clasificar familias de proteínas [9]. Por otro lado, la también creciente disponibilidad de datos estructurales han atraído a los interesados en la geometría computacional, los gráficos por ordenador y los algoritmos para analizar datos cristalográficos y procedentes de RMN (Resonancia Magnética Nuclear) y crear modelos moleculares verosímiles.

En muchos de estos campos, las técnicas de Soft Computing (ver sec-

ción 1.1) pueden jugar un papel crucial, al ser técnicas especialmente capacitadas para conseguir resultados de forma eficiente y con un buen nivel de calidad, además de permitir realizar modelizaciones que tengan en cuenta las imprecisiones o las carencias de información que pueden existir en los datos (debidas a conocimiento impreciso, a potenciales errores en las mediciones, etc.). En muchos casos, además, las soluciones óptimas de los modelos no son necesarias, pues puede haber mínimos (máximos) irrelevantes, y lo que interesa realmente es encontrar la solución biológicamente relevante. Así mismo, los SAD (ver sección 1.2) son una herramienta adecuada para el control centralizado de los diversos métodos existentes para un mismo problema y para el manejo de los grandes volúmenes de datos, de forma que se puedan validar de forma experimental los algoritmos, pues es este comportamiento práctico el aspecto más relevante. Igualmente, tanto los SAD como las técnicas de Soft Computing pueden ayudar en la generación e integración de información para dar lugar a conocimiento biológico nuevo.

Entre los muchos problemas a los que hace frente la Bioinformática, la tesis que aquí se presenta se enfoca a un problema propio del área de la Bioinformática estructural: la comparación de estructuras de proteínas. En las siguientes secciones veremos en más detalle las características de las estructuras de las proteínas y daremos una visión global de los métodos y modelos de comparación, resaltando aquéllos con una implicación más directa con esta tesis.

2.1. Fundamentos de la estructura de proteínas

La mayor parte de la estructura esencial de las células, y su función, conlleva la intervención de proteínas. Estás moléculas grandes y complejas muestran una notable versatilidad que les permite realizar un gran número de actividades fundamentales para la vida. De hecho, difícilmente podría cualquier otro tipo de macromolécula biológica asumir todas las funciones que las proteínas han acumulado durante billones de años de evolución. Cualquier mención de la función de una proteína debe estar soportada por la comprensión de la estructura de las proteínas, pues un principio fundamental en toda la ciencia Proteómica es que la estructura conduce a la función de la proteína. Las estructuras características de las proteínas permiten la colocación de grupos particulares de productos químicos en lugares

específicos del espacio tridimensional. Es esta precisión lo que permite que las proteínas actúen de catalizadores (enzimas) en una impresionante variedad de reacciones químicas. La colocación precisa de grupos de productos químicos también permite a las proteínas desempeñar importantes funciones estructurales, de transporte y regulatorias en los organismos.

Como la estructura de las proteínas conduce a su función, y las funciones de las proteínas son tan diversas, no sorprende descubrir que las estructuras de las proteínas son similarmente variadas [16]. Más aún, la diversidad de funciones de las proteínas se ve aumentada por la interacción de las mismas tanto con pequeñas moléculas como con otras proteínas.

Las décadas de investigación en el campo de las proteínas han producido una serie de principios acerca de la naturaleza de la estructura de las proteínas y la manera en que esta estructura se utiliza para llevar a cabo una función. Estos principios se han organizado en una jerarquía de cuatro capas que facilita la descripción y la comprensión de las proteínas: estructura primaria, secundaria, terciaria y cuaternaria. Dicha jerarquía no pretende describir de forma precisa las leyes físicas que producen la estructura de las proteínas, sino que más bien es una abstracción para hacer los estudios de estructuras de proteínas más manejables.

2.1.1. Estructura primaria de las proteínas: la secuencia de aminoácidos

Las proteínas son polímeros lineales de aminoácidos, y es la singular secuencia de aminoácidos constituyentes lo que determina la estructura tridimensional final de la proteína. La secuencia de aminoácidos se conoce como la estructura primaria de la proteína.

Los aminoácidos son pequeñas moléculas que contienen un grupo amino (NH_2), un grupo carboxilo (COOH), y un átomo de hidrógeno conectados a un carbono alfa central (C_α), como se puede ver en la figura 2.1. Además, los aminoácidos tienen una cadena lateral (o grupo R) conectado al carbono α, y es precisamente este grupo, y sólo este grupo, el que distingue un aminoácido de otro. Más aún, la cadena lateral es la que confiere las propiedades químicas específicas del aminoácido.

Los genomas celulares contienen instrucciones codificadas para la producción de múltiples proteínas, y hay 20 aminoácidos que pueden ser in-

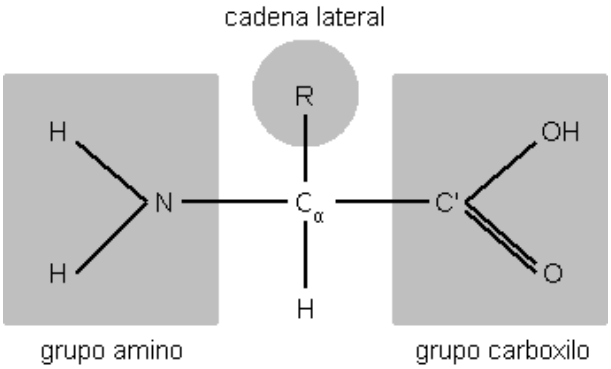


Figura 2.1: Estructura básica de un aminoácido.

corporados a una proteína mediante dichas instrucciones: Arginina (Arg), Alanina (Ala), Ácido aspártico (Asp), Asparagina (Asn), Cisteína (Cys), Glutamina (Gln), Ácido glutámico (Glu), Glicina (Gly), Histidina (His), Isoleucina (Ile), Leucina (Leu), Lisina (Lys), Metionina (Met), Fenilalanina (Phe), Prolina (Pro), Serina (Ser), Treonina (Thr), Triptófano (Trp), Tirosina (Tyr) y Valina (Val). La secuencia resultante de una proteína puede contener cualquier combinación de estos 20 aminoácidos, en cualquier orden. Aunque se sabía que los aminoácidos eran bloques para la construcción de proteínas antes del inicio del siglo veinte, el conjunto exacto de aminoácidos que se usan en las proteínas no se determinó hasta 1940 [42]. Este conjunto de 20 aminoácidos se considera estándar en el sentido de que es común a todos los organismos observados. Aunque existen formas modificadas de estos 20 aminoácidos en las proteínas, éstas son el producto de modificaciones que ocurren tras la síntesis de la proteína.

Los aminoácidos pueden formar enlaces entre sí a través de una reacción de sus respectivos grupos carboxilo y amino. El enlace resultante es llamado enlace peptídico, y se da el nombre de péptido a dos o más aminoácidos enlazados por este tipo de enlace (figura 2.2).

Una proteína se sintetiza por la formación de una sucesión lineal de enlaces peptídicos entre aminoácidos (siguiendo las instrucciones del código genético) y puede, por tanto, ser nombrada como un polipéptido. Cuando un aminoácido se incorpora en un péptido, se le denomina también como residuo, y los átomos envueltos en el enlace peptídico se conocen como es-

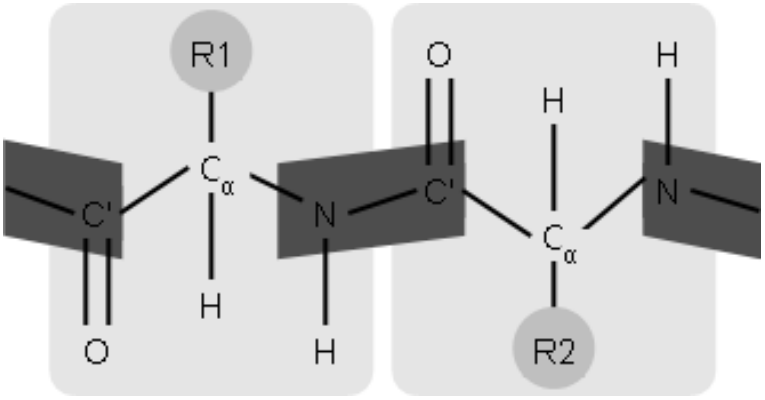


Figura 2.2: Enlace peptídico. Los dos aminoácidos marcados en gris claro se unen mediante el enlace peptídico indicado en gris oscuro. Los grupos R representan cualquiera de las 20 posibles cadenas laterales de los aminoácidos.

queleto péptido (peptide backbone). Las características específicas del enlace peptídico tienen importantes implicaciones para las estructuras tridimensionales que pueden formarse por polipéptidos. El enlace peptídico es planar (forma un plano), y es muy rígido, con lo que la cadena polipéptida tiene libertad de rotación solamente sobre de los enlaces formados por los carbonos α. No obstante, la libertad de rotación sobre los ángulos Φ ($C_\alpha-$ N) y Ψ ($C_\alpha-$ C') que conforman el enlace (ver figura 2.2) está limitada por la restricción que impone la distribución espacial de las cadenas laterales de los residuos y el esqueleto péptido, lo que hace que las posibles estructuras de un polipéptido concreto sean bastante limitadas.

2.1.2. Estructura secundaria de las proteínas

La estructura secundaria de una proteína puede ser vista como la estructura local de una cadena polipéptida, independientemente del resto de la proteína. Las limitaciones impuestas en la estructura primaria por el enlace peptídico y consideraciones relativas a puentes de hidrógeno dictan qué estructura secundaria es posible. Durante la investigación en estructura de proteínas, dos tipos de estructuras han sobresalido como las estructuras locales dominantes de cadenas polipéptidas: las hélices alfa (α helixes) y las láminas beta (β sheets). Estas estructuras exhiben un alto grado de regularidad: las combinaciones concretas de ángulos Φ y Ψ se repiten de forma

aproximada durante la duración de toda la estructura secundaria.

Aunque las hélices y las láminas satisfacen las restricciones de los enlaces peptídicos, este no es el único factor que explica su enorme presencia. Ambos elementos estructurales están estabilizados por las interacciones de los puentes de hidrógeno entre los átomos del esqueleto de los residuos participantes, haciendo que sean conformaciones altamente favorables para la cadena polipéptida. Las hélices y las láminas son los únicos elementos estructurales regulares presentes en las proteínas. Sin embargo, también se observa en las proteínas la presencia de elementos estructurales irregulares, y dichos elementos son vitales tanto para la estructura como para la función.

Hélices α

Las hélices se crean curvando el esqueleto polipéptido de tal forma que se produce una espiral regular. Debido a que el esqueleto polipéptido puede enrollarse en espiral en dos direcciones (izquierda o derecha), las hélices presentan dos posibles configuraciones. Una hélice con una espiral a mano derecha se denomina hélice a derechas (o hélice dextrógira), ya que vista desde arriba se observa que avanza en el sentido de las agujas del reloj. Este tipo de hélices es el predominante en las proteínas, dado que las restricciones espaciales limitan la capacidad de formación de hélices a izquierdas. Entre las hélices a derechas, las hélices α (ver figura 2.3) es también la más predominante. Una hélice α se caracteriza por tener un periodo de 3.6 residuos por vuelta del esqueleto en espiral. La estructura de esta hélice se estabiliza por interacciones de puentes de hidrógeno entre los oxígenos carbonilos (oxígenos del grupo carboxilo) de cada residuo y el protón del grupo amino del residuo que se encuentra 4 posiciones más adelante en la hélice. De esta forma, en la hélice α se satisfacen todos los enlaces posibles de hidrógenos, a excepción de unos pocos en cada extremo de la hélice donde no hay un compañero disponible.

Láminas β

Al contrario que las hélices, las láminas β (también llamadas hojas β o β sheets) están formadas por enlaces de hidrógeno entre cadenas de polipéptidos adyacentes en lugar de dentro de una única cadena (ver figura 2.4). Las secciones de la cadena de polipéptidos que participan en la lámina se co-

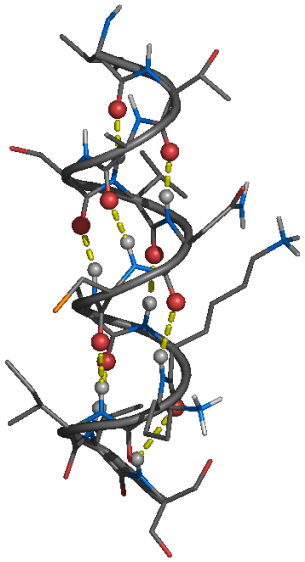
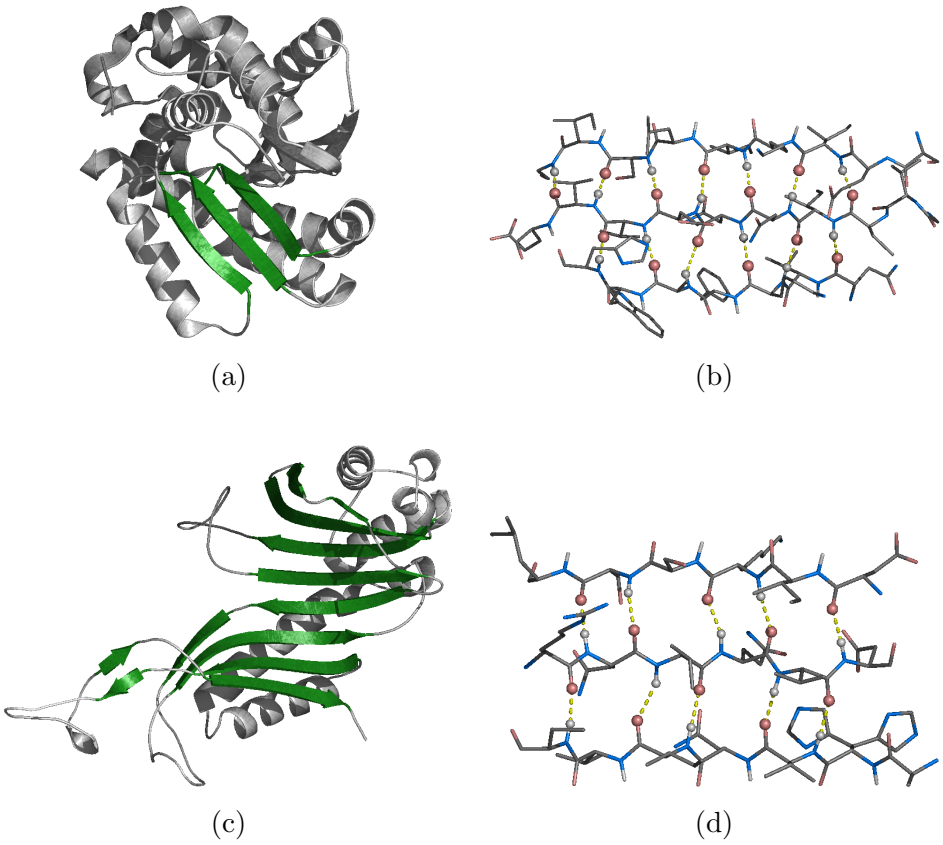


Figura 2.3: Hélice α.

nocen como hebras β (β strands). Las hebras β representan una estructura extendida de la cadena polipéptica, donde los ángulos Φ y Ψ están rotados aproximadamente 180 grados entre sí. Esta configuración produce una hoja β que está plegada, con las cadenas laterales de los residuos alternando posiciones en los lados opuestos de la hoja. Hay dos configuraciones posibles de láminas β: paralelas y antiparalelas. En las láminas paralelas, las hebras se alinean en la misma dirección con respecto a sus extremos amino (N) y carboxilo (C). En las láminas antiparalelas, las hebras alternan sus extremos amino y carboxilo, de tal forma que una hebra dada interacciona con hebras en la orientación contraria. Las láminas β pueden presentarse también en una configuración mixta, con secciones paralelas y antiparalelas, pero esta configuración es menos común que los dos tipos uniformes. Casi todas las láminas β presentan algún grado de torsión cuando la lámina se mira por el lado del filo, a lo largo de un eje perpendicular a la dirección de las cadenas polipéptidas, siendo esta torsión siempre a mano derecha.

Las hélices α y las láminas β suman la mayoría de las estructuras secundarias observadas en las proteínas. Estas estructuras regulares están intercaladas con regiones de estructuras irregulares llamadas giros o espirales. Las regiones de lazos se presentan generalmente en la superficie de la



(a) (b)

(c) (d)

Figura 2.4: Láminas β paralelas (a y b) y antiparalelas (c y d).

proteína. Estas regiones son a menudo simplemente transiciones entre estructuras regulares, pero también poseen significado estructural, y pueden ser la localización de la parte funcional o sitio activo de la proteína. Debido a su irregularidad, estos elementos son difíciles de clasificar. Sin embargo, algunos tipos de estructura son suficientemente frecuentes como para haber sido categorizados de manera aproximada. Los lazos en horquilla (hairpin loops o giros inversos) ocurren a menudo entre hebras beta antiparalelas e involucran el mínimo número de residuos (4-5) que se requieren para empezar la siguiente hebra. Existen muchos otros tipos de giros, que pueden clasificarse según las estructuras regulares que conectan y los ángulos Φ y Ψ de los residuos involucrados en el giro. Por ejemplo, las transiciones que involucran más residuos (6-16) son llamadas giros (o lazos) omega (Ω) porque se asemejan a la forma que tiene la letra griega omega mayúscula, y pueden conllevar interacciones complejas que incluyen a las cadenas laterales además del esqueleto péptido.

Aunque las regiones de giro son irregulares, estas estructuras están generalmente tan bien organizadas como las estructuras secundarias regulares. Sin embargo, la determinación experimental de estructuras de proteínas muestra que algunas regiones de giro no están tan bien organizadas y no alcanzan, por tanto, una estructura estable, recibiendo el nombre de espirales aleatorias (random coils).

2.1.3. Estructura terciaria de las proteínas: La estructura tridimensional global

La estructura terciaria de una proteína se define como la estructura tridimensional global de su cadena polipéptida. Hay una amplia variedad de formas en que los varios elementos de hélices, láminas y giros pueden combinarse para producir una estructura completa. Los puentes de hidrógeno, que ya desempeñan un papel en la estabilización de las estructuras secundarias, también son importantes en la estabilización de las estructuras terciarias, pero también son importantes otras fuerzas. En general, las combinaciones finales de estructuras secundarias están provocadas en gran medida por las interacciones entre las cadenas laterales de los residuos aminoácidos constituyentes de la proteína. Por tanto, al nivel de la estructura terciaria, las cadenas laterales juegan un papel mucho más activo en la creación de la estructura final, así como las interacciones del esqueleto eran las principales

responsables de la generación de estructuras secundarias (particularmente en el caso de hélices y láminas). De hecho, la mayoría de las proteínas tienen una estructura terciaria distintiva: los elementos de estructura secundaria de dichas proteínas se plegarán siempre de la misma forma para producir la misma estructura terciaria. Esta consistencia es vital para el funcionamiento adecuado de la proteína [16]. No obstante, hay casos de proteínas que no tienen una organización estructurada intrínseca, y que pasan mucho tiempo en estado desorganizado. La función de estas proteínas suele depender de alcanzar una estructura ordenada al darse ciertas condiciones o durante su participación en interacciones específicas [138].

El plegamiento de la proteína

La estructura terciaria tridimensional de una proteína se indica comúnmente como su plegamiento (fold). El plegamiento de una proteína (protein folding) es un proceso complejo que no se comprende completamente, pero se sabe que en la mayoría de los casos la estructura primaria de la proteína contiene toda la información requerida para que adquiera su plegado correcto. Más aún, una secuencia de polipéptidos aleatoria no se plegará casi nunca en una estructura organizada, siendo una propiedad importante de las secuencias de las proteínas el hecho de que se han seleccionado mediante el proceso evolutivo para alcanzar una estructura reproducible y estable [120]. A pesar de la naturaleza determinista del plegado de las proteínas, el proceso concreto de plegado es aún bastante desconocido, y no ha sido posible predecir de forma precisa la estructura final de una proteína dada solamente su secuencia [6, 16], aunque se van desarrollando métodos para afrontar este problema. Esto se debe en buena medida a que el número de estructuras tridimensionales conocidas de forma precisa era pequeño, pero a medida que este número aumente comenzará a ser posible desarrollar más y mejores modelos o deducir el plegamiento de una proteína en base al conocimiento del plegamiento de otras proteínas con secuencias similares (modelado por homología).

Dominios y motivos

Dentro de los plegamientos de proteínas existentes, se pueden reconocer diferentes dominios (domains) y motivos (motifs). Los dominios son secciones compactas de las proteínas que representan regiones independientes es-

tructuralmente (y a menudo también funcionalmente). Dicho de otro modo, un dominio es una subsección de una proteína que mantendría su estructura característica incluso si se separara del resto de la proteína. Los motivos (también llamados estructuras supersecundarias) son pequeñas subestructuras que no son necesariamente independientes estructuralmente, y que generalmente consisten de sólo unos pocos elementos estructurales secundarios. Se ven repetidamente motivos concretos en muchas estructuras de proteínas, y estos motivos constituyen los elementos integrantes de los plegamientos. Los motivos también tienen frecuentemente una importancia en la función y, en estos casos, representan unidades funcionales mínimas dentro de la proteína. Lógicamente, varios motivos pueden combinarse para la formación de un dominio específico.

Alteraciones de las proteínas

Aunque los aminoácidos tienen una gran variedad de características químicas, las alteraciones químicas y la interacción con moléculas no polipéptidas puede extender aún más las capacidades de las proteínas. Al nivel de la estructura terciaria es importante hacer mención de este fenómeno, ya que puede ser crítico tanto para la estructura como para la función de las proteínas [16].

Las propiedades químicas de los 20 aminoácidos estándar pueden extenderse mediante la modificación de las cadenas laterales y, en algunos casos, dicha modificación puede ser indispensable para la formación apropiada de las estructuras terciarias. Por ejemplo, el colágeno, una proteína fibrosa, contiene un residuo modificado de prolina, la hidroxiprolina, que estabiliza enormemente el plegamiento de la proteína [110]. En otros casos, la alteración de un residuo no tiene efecto en el plegamiento, sino que sirve para extender el repertorio funcional de la proteína.

Algunas moléculas, como los carbohidratos y los lípidos, pueden conectarse a la proteína mediante enlaces covalentes con residuos específicos. Las modificaciones debidas a carbohidratos se ven frecuentemente en proteínas que funcionan extracelularmente y se sabe que juegan un papel en la localización de proteínas intracelulares. Las modificaciones debidas a lípidos pueden ayudar a anclar la proteína en la membrana celular.

Las proteínas también pueden asociarse con pequeñas moléculas o átomos metálicos (de manera covalente o no covalente) diversificando una vez

más las capacidades funcionales y estructurales. Muchas enzimas emplean tales moléculas como cofactores, que las ayudan en la catálisis química [72]. Otras proteínas requieren estas moléculas o átomos para la correcta formación de su estructura terciaria. Por ejemplo, el motivo "dedo de zinc" (zinc-finger), un motivo estructural importante para la interacción de algunas proteínas con el ADN, no puede formarse sin la interacción covalente de un átomo de zinc con residuos específicos de cisteína o cisteína e histidina [84].

2.1.4. Estructura cuaternaria de las proteínas: Asociaciones de múltiples cadenas polipéptidas

La estructura terciaria de las proteínas describe la organización estructural de una única cadena polipéptida. Sin embargo, hay muchas proteínas que no funcionan como una única cadena o monómero, sino que existen en la forma de asociaciones no covalentes entre dos o más cadenas de polipétidos plegadas de forma independiente. Estas proteínas se denominan proteínas multisubunidades o proteínas multiméricas, y se dice que tienen una estructura cuaternaria. Las subunidades o protómeros pueden ser iguales o distintas, dando lugar, respectivamente, a proteínas homoméricas o heteroméricas [16].

La mayoría de las proteínas se pliegan de forma que se evita la agregación con otros polipéptidos, con lo que la formación de proteínas multiméricas es un tipo de interacción muy específica, aunque se estabiliza por el mismo tipo de interacciones que se emplean en la estabilización de las estructuras secundarias y terciarias. La creación de estructuras cuaternarias se explica por ventajas funcionales como puede ser ser la cooperación, la localización conjunta de funciones, la versatilidad en cuanto a funciones y capacidad de regulación, y el hecho de que simplifican la creación de estructuras de proteínas muy grandes al construirse a partir de subunidades más pequeñas.

2.1.5. Bases de datos de proteínas: PDB, CATH y SCOP

Existen varias bases de datos de proteínas que son accesibles públicamente. Entre ellas, las más destacables son el Protein Data Bank (PDB), que contiene ficheros con las coordenadas en 3D de los átomos de multitud de proteínas, y las bases de datos CATH y SCOP, con clasificaciones de las proteínas en base a distintos niveles de similaridad estructural.

Protein Data Bank (PDB)

El Protein Data Bank (PDB) [13, 14] es un repositorio de coordenadas e información relacionada para más de 38000 estructuras, incluyendo proteínas, ácidos nucleicos y grandes complejos macromoleculares. Estas coordenadas han sido determinadas de forma tridimensional (3D) haciendo uso de técnicas de cristalografía de Rayos X, resonancia magnética nuclear (NMR) y microscopía de electrones. Se trata de una base de datos internacional, gratuita y públicamente accesible, que durante muchos años desde su creación fue mantenida en exlusiva por el Research Collaboratory of Structural Bioinformatics (RSCB) en http://www.rcsb.org/pdb/.

Cada estructura del PDB lleva asignado un identificador de 4 caracteres alfanuméricos (llamado PDB ID o PDB code) y asociado a ese PDB ID existe un fichero con la información estructural de la misma. Es usual también, que cuando una estructura contiene varias cadenas, se identifique cada cadena con un caracter adicional, que suele concatenarse con el PDB ID cuando se quiere hacer referencia a dicha cadena de forma única. Los ficheros que conforman la base de datos PDB son ficheros de texto que contienen las coordenadas de los átomos de las estructuras a las que se refieren, así como comentarios con información adicional como puede ser información sobre la secuencia, los investigadores que definieron la estructura o información necesaria para la correcta comprensión de la información presente en el fichero. Un ejemplo de fichero PDB, para la proteína con PDB ID 1AA9, puede verse en la figura 2.5.

En la actualidad, y tras la creación del Worldwide Protein Data Bank (wwPDB) [13], del que forman parte el propio RCSB (EEUU), el MSD-EBI (Europa) y el PDBj (Japón), el RCSB mantiene solamente una copia de referencia estática del PDB (ftp://ftp.rcsb.org) que fue congelada con los datos disponibles a 31 de julio de 2007, y en ftp://ftp.wwpdb.org se mantiene una copia de producción, gestionada por el wwPDB (http://www.wwpdb.org/). Esta nueva copia va incorporando las actualizaciones del wwPDB Remediation Project, un proyecto destinado a corregir las inconsistencias detectadas en el repositorio y a conseguir que los datos del PDB se mantengan como una copia única consistente y en formatos uniformes, además de proporcionar los ficheros en más formatos como, por ejemplo, XML. No obstante, el wwPDB es muy reciente, y los experimentos previos realizados por otros

```
HEADER    PROTO-ONCOGENE                          27-JAN-97   1AA9
TITLE     HUMAN C-HA-RAS(1-171)(DOT)GDP, NMR, MINIMIZED AVERAGE
TITLE    2 STRUCTURE
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: C-HA-RAS;
COMPND   3 CHAIN: NULL;
COMPND   4 FRAGMENT: RESIDUES 1 - 171;
COMPND   5 ENGINEERED: SYNTHETIC GENE;
COMPND   6 OTHER_DETAILS: COMPLEXED TO GUANOSINE 5'-DIPHOSPHATE
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: HOMO SAPIENS;
SOURCE   3 ORGANISM_COMMON: HUMAN;
SOURCE   4 EXPRESSION_SYSTEM: ESCHERICHIA COLI;
SOURCE   5 EXPRESSION_SYSTEM_STRAIN: TG1;
SOURCE   6 EXPRESSION_SYSTEM_PLASMID: PRGH;
SOURCE   7 EXPRESSION_SYSTEM_GENE: HUMAN C-HA-RAS GENE
KEYWDS    RAS, ONCOGENE PROTEIN, GTP-BINDING PROTEIN, PROTO-ONCOGENE
EXPDTA    NMR, MINIMIZED AVERAGE STRUCTURE
AUTHOR    Y.ITO,Y.YAMASAKI,Y.MUTO,G.KAWAI,S.NISHIMURA,T.MIYAZAWA,
AUTHOR   2 S.YOKOYAMA
REVDAT   1   29-JUL-97 1AA9    0
REMARK   1
REMARK   2
REMARK   2 RESOLUTION. NOT APPLICABLE.
...
ATOM      1  N   MET     1     129.640  14.669 -14.847  1.00  2.13           N
ATOM      2  CA  MET     1     130.624  13.884 -14.049  1.00  1.42           C
ATOM      3  C   MET     1     130.508  12.401 -14.404  1.00  1.16           C
ATOM      4  O   MET     1     129.625  11.993 -15.134  1.00  1.52           O
ATOM      5  CB  MET     1     130.340  14.070 -12.561  1.00  0.96           C
ATOM      6  CG  MET     1     131.298  15.114 -11.985  1.00  1.06           C
ATOM      7  SD  MET     1     130.359  16.569 -11.458  1.00  1.78           S
ATOM      8  CE  MET     1     130.352  17.423 -13.053  1.00  2.41           C
ATOM      9 1H   MET     1     129.044  14.019 -15.399  1.00  2.33           H
ATOM     10 2H   MET     1     129.043  15.230 -14.208  1.00  2.50           H
ATOM     11 3H   MET     1     130.148  15.307 -15.492  1.00  2.68           H
ATOM     12  HA  MET     1     131.620  14.227 -14.265  1.00  1.63           H
ATOM     13 1HB  MET     1     129.322  14.400 -12.427  1.00  1.53           H
ATOM     14 2HB  MET     1     130.487  13.133 -12.051  1.00  1.20           H
ATOM     15 1HG  MET     1     131.819  14.695 -11.137  1.00  1.44           H
ATOM     16 2HG  MET     1     132.013  15.402 -12.741  1.00  1.47           H
ATOM     17 1HE  MET     1     130.217  16.701 -13.847  1.00  2.88           H
ATOM     18 2HE  MET     1     129.544  18.142 -13.072  1.00  2.81           H
ATOM     19 3HE  MET     1     131.290  17.937 -13.192  1.00  2.81           H
ATOM     20  N   THR     2     131.392  11.591 -13.891  1.00  0.74           N
ATOM     21  CA  THR     2     131.334  10.134 -14.195  1.00  0.55           C
ATOM     22  C   THR     2     130.993   9.362 -12.927  1.00  0.46           C
ATOM     23  O   THR     2     131.108   9.865 -11.830  1.00  0.48           O
...
```

Figura 2.5: Fragmento de fichero PDB para la proteína con código 1AA9.

autores usaban los datos originales en el RSCB PDB. Por tanto, para poder realizar comparaciones de forma fiable con otros métodos, y dado que el repositorio del wwPDB no ha aparecido hasta los momentos finales del desarrollo de esta tesis, en lo que sigue, al hacer referencia a un fichero del PDB nos estaremos refiriendo a las copias presentes en la versión de referencia estática que se mantiene aún en el RSCB.

CATH

La base de datos CATH [107, 111] es una base de datos de dominios estructurales de proteínas disponible en http://www.cathdb.info/ que en su última versión (enero de 2007) contiene 93885 estructuras de dominios. CATH realiza una clasificación jerárquica de los dominios, por medio de procedimientos tanto automáticos como manuales, agrupando las proteínas en 4 grandes niveles: Clase (Class - C), Arquitectura (Architecture - A), Topología (Topology - T) y superfamilia Homóloga (Homologous superfamily - H), de cuyas iniciales se deriva el nombre de CATH. En más detalle, las características de estos 4 niveles son:

- **Clase (C)**: La clase se determina de la composición en estructuras secundarias de la proteína y de la forma en que aparecen distribuidas en ella. Se reconocen 3 clases principales: mainly-alpha, mainly-beta y alpha-beta, incluyendo esta última tanto estructuras alternadas alpha/beta como estructuras de tipo alpha+beta. Además, una cuarta clase se asocia a las proteínas con poco contenido en estructuras secundarias.

- **Arquitectura (A)**: Este nivel describe la forma global del dominio determinada por las orientaciones de las estructuras secundarias, pero sin tener en cuenta la conexión entre ellas. Se asigna de una forma manual usando descripciones muy simples de la disposición estructural como “barril” o “emparedado de tres capas”.

- **Topología (T)**: Las estructuras se clasifican en grupos de plegamiento (fold groups) dependiendo tanto de su forma global como de la interconexión de sus estructuras secundarias. Esta clasificación se realiza mediante los algoritmos de comparación SSAP y CATHEDRAL, asignándose al mismo grupo las estructuras con puntuaciones de SSAP

de 70 y donde al menos el 60 % de la proteína más larga tiene correspondencia con la proteína más pequeña.

- **Superfamilia Homóloga (H)**: En este nivel se agrupan los dominios de proteínas que se piensa que tienen un ancestro común y que, por tanto, pueden ser descritos como homólogos. Las estructuras se clasifican en la misma superfamilia homóloga si satisfacen criterios apropiados en el porcentaje de identidad a nivel de secuencia, superposición alta de la estructura más larga con la más pequeña (alto nivel de equivalencia),...

SCOP

La base de datos Structural Classification of Proteins (más conocida como SCOP) [3,105] es otra base de datos con una clasificación, en gran medida manual, de dominios estructurales de proteínas, basada en las similaridades entre sus secuencias y sus estructuras tridimensionales. Los niveles presentes en SCOP, del nivel más bajo al más alto de la jerarquía son:

- **Especie (Species)**: Este nivel es el más específico y representa una secuencia distintiva de proteína y las variantes naturales o artificiales que pueda tener.
- **Proteína (Protein)**: Agrupa secuencias similares que desempeñan esencialmente las mismas funciones que pueden haberse originado en diferentes especies biológicas o ser isomorfismos presentes en un mismo organismo.
- **Familia (Family)**: Contiene proteínas con secuencias relacionadas pero cuyas funciones son típicamente distintas.
- **Superfamilia (Superfamily)**: Une las familias de proteínas con características comunes a nivel funcional y de estructura, provenientes probablemente de un ancestro común en su evolución.
- **Plegamiento (Fold)**: Agrupa superfamilias similares con algunas características propias distintivas.
- **Clase (Class)**: Este nivel se basa principalmente en el contenido en estructuras secundarias y su organización.

2.2. Comparación de estructuras de proteínas

A medida que el número de estructuras conocidas de proteínas 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 orígenes evolutivos comunes. En la actualidad, existe un consenso respecto a que las similaridades en proteínas distantes se preservan más a nivel de estructura tridimensional, aún cuando prácticamente no exista relación a nivel de secuencia de aminoácidos.

Desde el punto de vista algorítmico, la comparación de estructuras 3D de proteínas es un problema que supone un gran desafío. 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 nuevos medicamentos o drogas, la identificación de nuevos tipos de estructuras proteicas, y la organización y clasificación del universo conocido de estructuras de proteínas, a la vez que podrían contribuir en el descubrimiento de inesperadas interrelaciones evolutivas y funcionales entre ellas.

Más aún, la disponibilidad de buenas técnicas de comparación de estructuras de proteínas podría usarse también en la evalución de predicciones estructurales por threading o modelización por homología (homology modeling) ab-initio.

Puede afirmarse que la comparación de estructuras de proteínas, y la ulterior clasificación de las mismas (de acuerdo a su similaridad) es un aspecto fundamental en la investigación actual en campos importantes de la moderna Genómica y Proteómica estructural [69, 74, 79].

2.2.1. 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 or-

denados 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, dadas dos estructuras y una función de costo, permiten obtener la equivalencia de mejor costo. Sin embargo, el problema general es NP-completo, incluso en sus formulaciones más simples [50], 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:

$$RMSD_c = \sqrt{\frac{1}{r}\sum_{i=1}^{r}(a_i - b_i)^2} \tag{2.1}$$

donde $a_i \in A, b_i \in B$ se entienden como el conjunto de coordenadas asociadas al residuo i en cada proteína. Por lo general, este conjunto es unitario (un sólo "i" por residuo) y contiene las coordenadas del carbono central del residuo.

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

$$RMSD_d = \frac{1}{r}\sqrt{\sum_{i=1}^{r}\sum_{i=1}^{r}(d_{ij}^A - d_{ij}^B)^2} \tag{2.2}$$

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 ante 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 cuál de ellas representa la "correcta" no es una tarea simple.

Por otro lado, la utilización de medidas de similaridad estructural basadas en RMSD presenta el problema de la sensibilidad frente a la presencia de outliers. Existen otras formas de evaluar la similaridad que no serán reseñadas aquí por no ser éste el objeto central de esta tesis. El lector interesado puede referirse al trabajo de Kohel [74] y las referencias allí citadas, o al trabajo de May [95] donde se comparan hasta 37 medidas de similaridad. No obstante, en los experimentos de los capítulos 4 y 5, se usarán las medidas de similaridad propias de los métodos preexistentes utilizados y las nuevas medidas utilizadas en los métodos nuevos que han sido desarrollados.

En la actualidad se aplican varios tipos de estrategias y metodologías para la comparación de estructuras de proteínas [16, 36, 79], entre las que se incluyen los cliques maximales [43], programación dinámica [130], comparación de matrices de distancia [5, 68], teoría de grafos [126], hashing geométrico [85], análisis de correlación de componentes principales [139], algoritmos genéticos [129], alineamiento local y global [146], consensus shapes [24], consensus structures [86], comparación de las proteínas como caminos en 3D [147], extensión combinatoria de la trayectoria óptima [124],..., de los que describiremos alguno en las próximas secciones mientras que otros serán descritos de forma más breve en las descripciones de los experimentos desarrollados en esta tesis en los que intervengan.

Finalizaremos el capítulo entrando en más detalle en las características de los métodos basados en la complejidad de Kolmogorov (Medida Universal de Similaridad, Universal Similarity Metric o USM) [77], los mapas de contacto estándar [78,140,141] y los mapas de contacto difusos [112], pues es en estos últimos métodos en los que se centra el trabajo sobre comparación de estructuras de proteínas que se desarrolla en esta tesis. En concreto, en los capítulos 4 y 5, veremos la aplicación de la medida universal de similaridad sobre los mapas de contacto difusos y el uso de una técnica metaheurística basada en la búsqueda por entornos variables para ambos tipos de mapas de contactos. Hay que indicar, también, que los mapas de contacto son modelos en forma de grafo de una proteína, que abstraen el problema biológico, y que nos han permitido realizar una aproximación gradual al problema de comparación de proteínas, sin necesidad de un conocimiento exhaustivo previo

de Proteómica o Biología.

2.2.2. Comparación de estructuras vía Matrices de Distancia usando DALI

La utilización de matrices de distancia para la comparación de proteínas dió lugar a uno de los algoritmos más difundidos en la actualidad: DALI [68].

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:

$$S = \sum_{i=1}^{L} \sum_{j=1}^{L} \phi(i,j) \tag{2.3}$$

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 "elástica", que es tolerante al efecto acumulativo de distorsiones geométricas graduales, y cuya definición es:

$$\phi^E(i,j) = \begin{cases} \theta^E - \frac{|d_{ij}^A - d_{ij}^B|}{d_{ij}^*} w(d_{ij}^*), & \text{si } i \neq j \\ \theta^E & \text{si } i = j \end{cases} \tag{2.4}$$

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 más significativas.

DALI consta de dos pasos. Una vez construidas las matrices de distancia para ambas proteínas, el primer 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, es decir, se comparan todas las submatrices de tamaño 6×6. 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 similaridad. 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 encontraba accesible en http://www.ebi.ac.uk/dali, desde donde se podían ejecutar comparaciones estructurales, pero aunque dicho servicio fue un referente de los métodos de comparación de proteínas desde el año 1996, los cambios técnicos y de equipos en el European Bioinformatics Institute (EBI), en donde se alojaba, han conducido a su eliminación en enero de 2008. No obstante, aún se encuentra disponible una distribución (*DaliLite* [90]) que permite ejecutar el algoritmo en forma local, y que se puede obtener en http://ekhidna.biocenter.helsinki.fi/dali_lite/downloads/download.html. Esta versión de distribución también ha sido puesta online en un nuevo servicio web del EBI, en la dirección http://www.ebi.ac.uk/DaliLite/.

2.2.3. Comparación de estructuras vía Matrices de Distancia usando MatAlign

MatAlign, al igual que DALI, utiliza matrices de distancia como punto de partida para realizar la comparación de proteínas. Se trata de un algoritmo más reciente [5], que trata de emparejar residuos basándose en el hecho de que dos residuos estructuralmente equivalentes tendrán unos perfiles de distancia similares (los valores en su fila de la respectiva matriz de distancia serán similares).

El método usado por MatAlign difiere del utilizado en DALI, en el sentido de que no subdivide la matriz de distancia en matrices de 6×6. En su lugar, MatAlign hace uso de programación dinámica en dos niveles. En primer lugar, se hace un alineamiento de tipo fila-fila y, en segundo lugar, se usa programación dinámica para consolidar las puntuaciones de los pares fila-fila y producir un alineamiento inicial. Posteriormente, este alineamiento es refinado de forma iterativa para conseguir un alineamiento final haciendo uso de una función objetivo de puntuación del alineamiento que es propia del método.

MatAlign ha sido comparado en términos de precisión y velocidad contra DALI [90] y CE [124], mostrando mejores resultados que DALI y CE en una serie de experimentos, en los que superó a DALI en términos de velocidad, acercándose a los tiempos de CE [5].

2.2.4. 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 algorit-

mos que resuelvan el problema de *subgrafo común maximal* (MCS). En esta sección se describe el trabajo de Gardiner et. al. [43] donde se utiliza esta técnica para la detección de farmacóforos comunes en dos moléculas. Un enfoque usual para la detección de farmacóforos comienza con la identificación de estructuras que sean activas en algún aspecto de interés. Posteriormente 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 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 [7], 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 moléculas.

En el trabajo reseñado, los vértices 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 [22]. Además, sugieren que el algoritmo clásico de Bron-Kerbosch [17] es la opción a elegir si se desean obtener todos los cliques maximales en lugar de únicamente el máximo. Finalmente se indica que estas conclusiones son válidas para grafos de "baja" densidad, como los implicados por la representación utilizada. La densidad se calcula como $\frac{2k}{e(e-1)}$, siendo e el número de vértices y k el de aristas.

2.2.5. Comparación de estructuras vía extensión combinatoria de la trayectoria óptima

El método de comparación de estructuras mediante extensión combinatoria de la trayectoria óptima (CE) [124] se basa en la creación de trayectorias (caminos) entre pares de fragmentos alineados (AFP) correspondientes a fragmentos de ambas proteínas con similaridad estructural. En este método, el alineamiento obtenido entre dos proteínas A y B de tamaños respectivos n^A y n^B será el camino continuo más largo que recorre los AFP de tamaño m en una matriz de similaridad S de tamaño $(n^A - m) \cdot (n^B - m)$ con todos los posibles AFP que cumplen el criterio de similaridad estructural. Dos AFP i e $i+1$ consecutivos en la trayectoria de alineamiento deben cumplir una de las 3 condiciones siguientes:

$$p_{i+1}^A = p_i^A + m \ y \ p_{i+1}^B = p_i^B + m \tag{2.5}$$

ó

$$p_{i+1}^A > p_i^A + m \ y \ p_{i+1}^B = p_i^B + m \tag{2.6}$$

ó

$$p_{i+1}^A = p_i^A + m \ y \ p_{i+1}^B > p_i^B + m \tag{2.7}$$

donde p_i^A es la posición del residuo de la proteína A para el AFP en la posición i de la trayectoria de alineamiento y p_i^B se define de forma similar para la proteína B. La primera condición (2.5) describe 2 AFP consecutivos sin huecos entre ellos, y las otras 2 condiciones describen dos AFP consecutivos con huecos en la proteína A (2.6) y en la proteína B (2.7). La trayectoria de alineamiento se construye entre AFPs de tamaño fijo (típicamente se ha visto que un valor de 8 es un valor razonable). El primer AFP en el que comienza la trayectoria puede elegirse entre todos los AFP de la matriz S, mientras que los siguientes AFP se eligen de acuerdo a las condiciones 2.5-2.7, estableciéndose restricciones adicionales del tamaño máximo de los huecos (normalmente 30). El valor del tamaño máximo entre huecos es proporcional al tiempo de ejecución con lo que no se pueden usar valores muy grandes, pero también conlleva que se pierdan, o se representen mal, los alineamientos con huecos de mayor tamaño.

El algoritmo puede usar 3 estrategias principales distintas para la ampliación de la trayectoria de alineamiento:

1. Considerar todos los AFP que extienden la trayectoria cumpliendo los criterios exigidos.

2. Considerar sólo el mejor AFP.

3. Usar estrategias intermedias entre las dos anteriores.

La primera estrategia da lugar a una búsqueda combinatoria exhaustiva por la trayectoria óptima, mientras que la segunda da lugar a una búsqueda limitada entre los mejores caminos. Se ha comprobado que la segunda estrategia es suficiente para descubrir similaridades estructurales cuando se combina con heurísticas que evalúen el camino, a la vez que es considerablemente superior en cuanto a rendimiento. El algoritmo considera normalmente todos los posibles puntos de inicio para las trayectorias, pero se queda sólo con el alineamiento de longitud máxima entre todos los obtenidos. En cuanto a la elección de AFP a la hora de extender un camino, y la decisión de si una trayectoria debe ser extendida o no, se usan heurísticas basadas en la distancia entre los fragmentos alineados en los AFP.

Finalmente el algoritmo realiza un paso de mejora entre las mejores trayectorias obtenidas que contribuye a reducir el RMSD de las mismas.

2.2.6. Mapas de contacto

Hemos visto que DALI (sección 2.2.2) y MatAlign (sección 2.2.3) usaban matrices de distancia para realizar la comparación de las proteínas. Los mapas de contacto son un enfoque similar, aunque más sencillo, en los que entra en juego el concepto de contacto. De esta forma, dos elementos (aminoácidos, residuos o átomos) estarán en contacto si la distancia entre ellos está por debajo de cierto umbral, expresado generalmente en Angstroms(Å). Al aplicar el concepto de contacto, las matrices de distancia se transforman en matrices de valores discretos $\{0, 1\}$ con dimensión $n \times n$, donde n es el número de elementos en la proteína. Para cada elemento (i, j), la matriz contendrá un 1 si dichos elementos están en contacto o un 0 en caso contrario. La figura 2.6 muestra un ejemplo gráfico de este tipo de representación, donde los 0 en la matriz se dibujan en color blanco y los 1 en color negro.

Alternativamente, un mapa de contacto puede verse como un grafo donde los nodos representan los elementos de la proteína y dos elementos i, j estarán conectados por una arista ("contacto") si existe contacto entre ellos

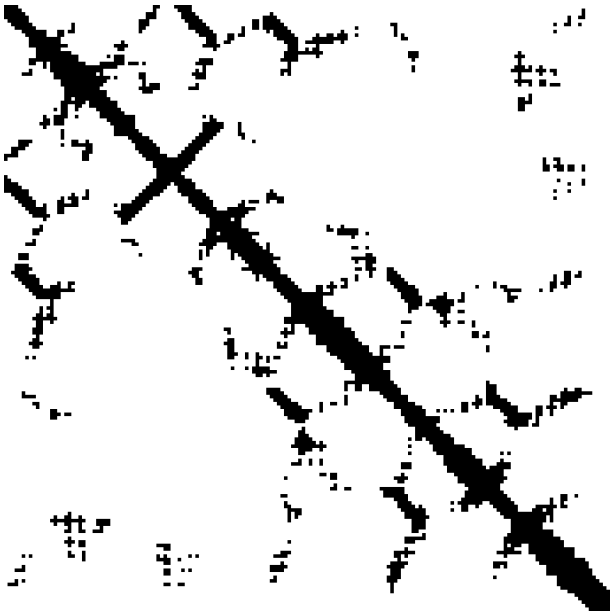


Figura 2.6: Representación mediante matriz del mapa de contacto de la proteína 1AA9.

(en el mismo sentido anterior de distancia por debajo de un valor de umbral). Se puede ver un ejemplo de la proteína 1LYZ, usando esta representación, en la figura 2.7. En la mayoría de las casos los elementos utilizados son los residuos, representados por sus carbonos alfa centrales (C_α) y, en general, en el resto de esta tesis, usaremos siempre residuos al referirnos a los componentes de un mapa de contacto, aunque debe entenderse que podría ser también cualquier otro de los mencionados elementos.

El concepto de mapas de contacto puede ser generalizado mediante los mapas de contacto difusos [112], en los que no sólo se contempla la existencia o no de contacto entre dos residuos, sino que dichos contactos llevarán asociado un tipo. Así se logra permitir que se capte mejor la semántica de los contactos, al distinguir, por ejemplo, entre contactos cortos (muy pocos Ansgtroms de distancia) y largos (una distancia en Angstroms moderada), tal y como se puede ver en la figura 2.8. La utilización de distintas funciones de pertenencia está motivada, pues, por el hecho de que una proteína puede presentar simultáneamente diferentes patrones de contacto. Por ejemplo, la distancia media entre residuos en contacto pertenecientes a una hélice α es

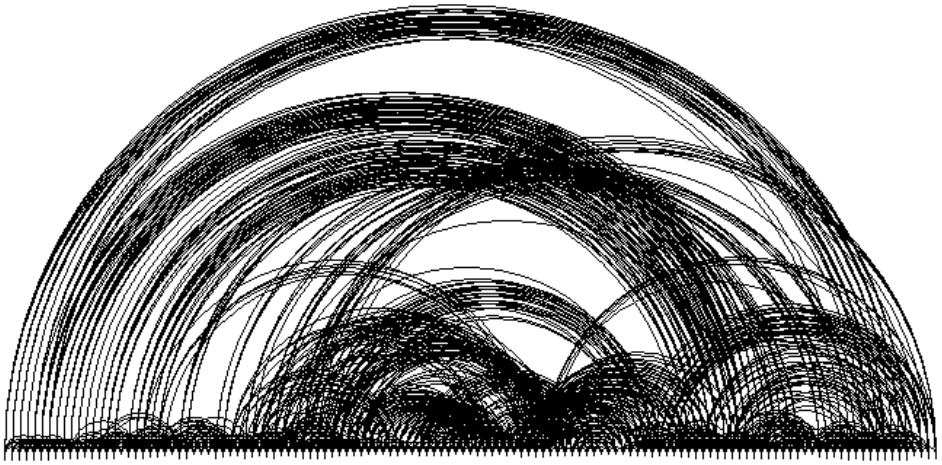


Figura 2.7: Representación mediante grafo del mapa de contacto de la proteína 1LYZ.

diferente a la distancia media entre residuos conectados de una lámina β. Además, existen errores potenciales en la determinación de las coordenadas en 3D de los átomos de una proteína, con lo que la validez de un valor de umbral único y preciso es cuestionable. En [81] se estableció que los errores experimentales en la determinación de las coordenadas atómicas cartesianas por Cristalografía de Rayos X o NMR varían desde 0.01 a 1.27Å, lo que está cerca del valor de algunos enlaces covalentes. Por tanto, la utilización de mapas de contacto difuso tiene sentido también para tratar con estos potenciales errores e imprecisiones, relajando la distancia exigida para considerar que existen contactos.

Para la definición de los tipos de contacto puede hacerse uso de un conjunto de funciones de pertenencia, de forma que cada función representará un tipo de contacto difuso distinto, dependiendo de la distancia en Angstroms que haya entre los residuos. Si se numeran los tipos de contacto difuso de forma creciente, empezando con el valor 1, el tipo asignado a un contacto será el valor numérico del contacto difuso en cuya función de pertenencia obtiene un grado de pertenencia más alto. No obstante, si el grado de pertenencia a todos los tipos de contacto definidos es 0, se considerará que el tipo de contacto es también 0. En la figura 2.9 se pueden ver tres ejemplos de conjuntos de funciones de pertenencia. El primero de los conjuntos

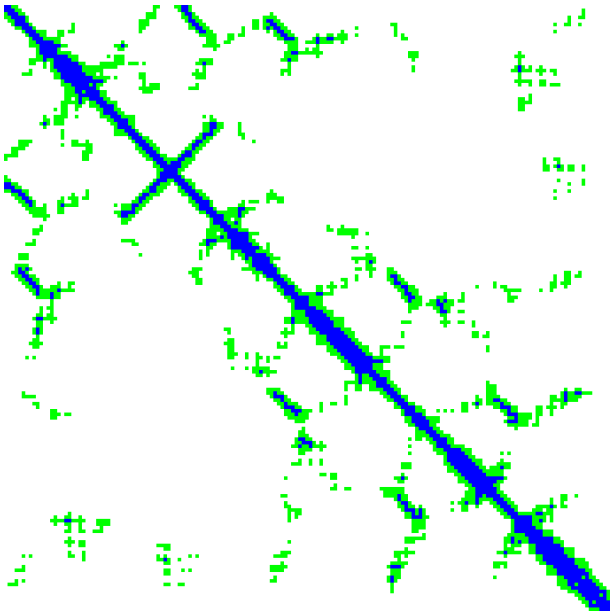
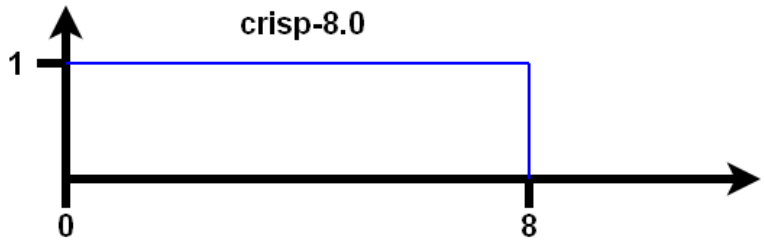
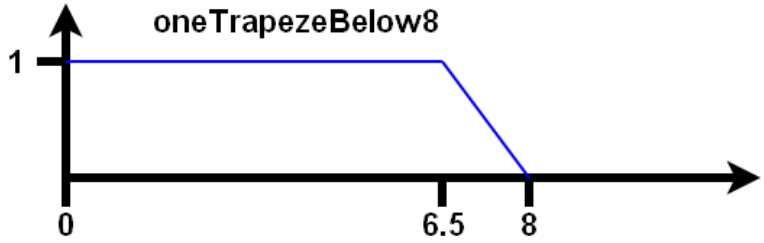


Figura 2.8: Representación mediante matriz del mapa de contacto difuso de la proteína 1AA9 para dos tipos de contactos: cortos y largos.

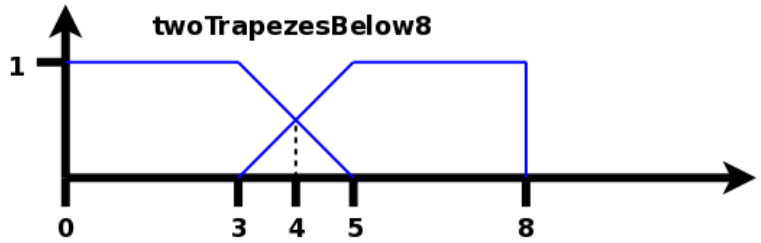
de funciones (a) sólo contiene una función de pertenencia, que establece la existencia de contacto, con grado de pertenencia 1, para residuos a distancias entre 0 y 8Å, de forma que si se utilizara dicho conjunto de funciones para crear un mapa de contacto difuso se obtendría el equivalente a un mapa de contacto normal con un umbral de 8Å. El segundo conjunto de funciones (b) contiene una función de pertenencia que produciría exactamente los mismos contactos que un mapa de contacto normal con umbral de 8Å, pero con la diferencia de que los valores de pertenencia ya no son todos 1, sino que entre 6.5 y 8Å se produce un descenso gradual del valor de pertenencia, o, lo que es lo mismo, disminuye paulatinamente el grado de pertenenencia al tipo de contacto para los residuos a distancia mayor de 6.5Å hasta que, a partir de 8Å, el grado de pertenencia es 0. Por último, el tercer conjunto de funciones (c) contiene dos funciones de pertenencia, dando como resultado que los residuos a distancias entre 0 y 4Å se considerará que tienen un contacto de tipo 1 (primera función, *contacto corto*) y los residuos a distancias entre 4 y 8Å tendrán un contacto de tipo 2 (segunda función, *contacto largo*).



(a)



(b)



(c)

Figura 2.9: Tres conjuntos de funciones de pertenencia utilizables para crear mapas de contacto difusos.

2.2.7. USM: Comparación de mapas de contactos mediante la medida universal de similaridad

La similaridad de dos mapas de contacto (difusos o no) puede ser evaluada mediante la medida universal de similaridad (USM), que aproxima cualquier otra posible medida de similaridad [87]. La clave de la medida universal de similaridad radica en el concepto de complejidad de Kolmogorov $K(.)$ de un objeto o, definido como la longitud del programa más corto que necesita una máquina universal de Turing U para producir o como salida. Siguiendo [88] tenemos:

$$K(o) = \text{mín}\{|P|,\ P \textit{ es un programa y } U(P) = o\} \tag{2.8}$$

Una medida relacionada es la complejidad condicional de Kolmogorov de o_1 dado o_2:

$$K(o_1|o_2) = \text{mín}\{|P|,\ P \textit{ es un programa y } U(P, o_2) = o_1\} \tag{2.9}$$

La ecuación 2.9 mide cuánta información se necesita para producir el objeto 1 si conocemos el objeto 2. De este modo, la distancia en información (Information Distance o ID), entre dos objectos, de acuerdo con [12], es equivalente (hasta el nivel de un término de tipo sumando logarítmico) a:

$$ID(o_1, o_2) \quad = \quad \text{máx}\{K(o_1|o_2), K(o_2|o_1)\} \tag{2.10}$$

La medida universal de similaridad (tal y como aparece en [87]) es una métrica adecuada, universal y además normalizada. Se define como:

$$d(o_1, o_2) \quad = \quad \frac{\text{máx}\{K(o_1|o_2^*), K(o_2|o_1^*)\}}{\text{máx}\{K(o_1), K(o_2)\}} \tag{2.11}$$

donde o_i^* indica el programa más corto para o_i.

Desafortunadamente, $K(0)$ es no-computable, con lo que se requiere un modo de aproximar $K(o)$ y $K(o_i|o_j)$.

Afortunadamente, en un trabajo previo del uso de USM para comparar mapas de contacto [77], se introdujo una aproximación adecuada, tomando

como guía a [88], que consistía en representar cada mapa de contacto como una cadena s y aproximar $K(s)$ como el tamaño (en número de bytes) de la cadena comprimida $zip(s)$, es decir, $K(s) \approx |zip(s)|$. La cadena s no es más que la representación textual de la matriz del mapa de contacto.

El cálculo del término $K(o_1|o_2)$ se hacía mediante $K(o_1 \cdot o_2) - K(o_2)$ donde $\cdot$ denota concatenación de cadenas y $K(.)$ se estimaba de la forma recién explicada.

En esta tesis se extenderá el trabajo de [77] en lo referente a USM estudiando cómo el uso de mapas de contacto difusos junto a la medida universal de similaridad (ecuación 2.11) puede mejorar los resultados frente al uso de mapas de contacto estándar. Dicha extensión se presenta en la sección 5.1 del capítulo 5.

2.2.8. Max-CMO: Problema de la máxima superposición de mapas de contacto

El problema de la máxima superposición de mapas de contacto (Max-CMO) es un modelo matemático que permite comparar la similaridad de dos estructuras de proteínas tomando como base los mapas de contacto de las mismas. Una instancia del Max-CMO es, por tanto, un par de mapas de contacto de las proteínas a comparar, y el objetivo es construir el alineamiento entre dichos mapas que maximice un cierto coste. Un alineamiento indica una correspondencia entre los elementos (aminoácidos, residuos o átomos) de ambas proteínas, de forma que quedan emparejados aquellos elementos que se consideran equivalentes.

El valor de una superposición de dos mapas de contacto (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, formando de esta forma ciclos de longitud 4 (4 aristas). Un ejemplo se muestra en la figura 2.10, donde los residuos *1, 2, 3* y *6* de la proteína inferior (P_1) están emparejados con los residuos *2, 3, 5* y *7* de la proteína superior (P_2). El alineamiento está representado en la figura por líneas de puntos, mientras que los contactos entre los residuos de las proteínas se muestran con líneas continuas.

Este alineamiento particular produce dos ciclos de longitud 4. El primer ciclo se compone de las aristas $(1 \in P_1, 3 \in P_1)$, $(3 \in P_1, 5 \in P_2)$, $(5 \in P_2, 2 \in P_2)$ y $(2 \in P_2, 1 \in P_1)$. El segundo ciclo tiene las siguientes 4 aristas:

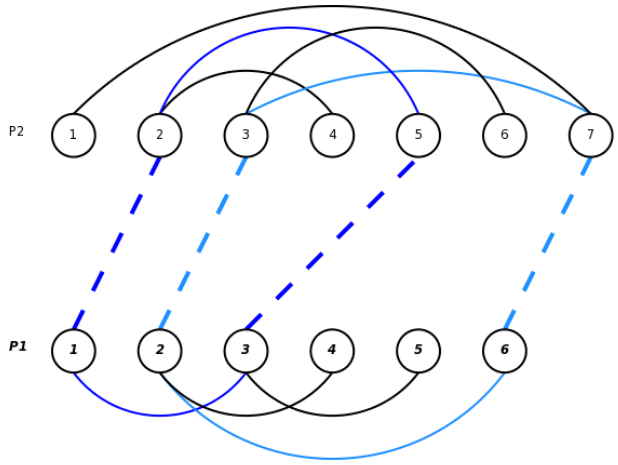


Figura 2.10: Ejemplo de un alineamiento entre dos proteínas. El valor de superposición (valor objetivo de la solución) es 2 porque en la solución aparecen dos ciclos de longitud 4. El primer ciclo se compone de las aristas $(1 \in P_1, 3 \in P_1)$, $(3 \in P_1, 5 \in P_2)$, $(5 \in P_2, 2 \in P_2)$ y $(2 \in P_2, 1 \in P_1)$. El segundo ciclo tiene las siguientes 4 aristas: $(2 \in P_1, 6 \in P_1)$, $(6 \in P_1, 7 \in P_2)$, $(7 \in P_2, 3 \in P_2)$, y $(3 \in P_2, 2 \in P_1)$.

$(2 \in P_1, 6 \in P_1)$, $(6 \in P_1, 7 \in P_2)$, $(7 \in P_2, 3 \in P_2)$, y $(3 \in P_2, 2 \in P_1)$. Si denominamos $\sigma()$ a la función que para cada elemento de P_1 nos da su pareja en P_2 (si tiene una en el alineamiento considerado), un ciclo es una tupla de 4 elementos $(i, j, \sigma(i), \sigma(j))$.

Si se plantea el Max-CMO como un problema de optimización, lo que se busca es maximizar la cantidad de ciclos de longitud 4. Además, se establece una restricción adicional que evita los cruces entre aristas, de forma que para todo par de aristas $(i \in P_1,\ \sigma(i) \in P_2)$, $(j \in P_1,\ \sigma(j) \in P_2)$, se cumplirá que si $i < j$ entonces $\sigma(i) < \sigma(j)$. Esta restricción tiene su fundamento en que los residuos equivalentes entre dos proteínas normalmente se presentarán en grupos, asociados a trozos similares entre ambas proteínas, careciendo de sentido biológico el establecimiento de equivalencias entre conjuntos de residuos muy dispersos. Este problema es NP-Completo [50, 75] y en [78] se presentó el primer algoritmo exacto para resolverlo, extendido posteriormente en [20,21,76]. Un nuevo algoritmo exacto ha sido presentado más recientemente en [140,141]. Ambos algoritmos exactos serán explicados brevemente a continuación y veremos por qué, pese a la existencia de es-

tos algoritmos, sigue siendo necesario disponer de técnicas heurísticas para resolver el problema.

El primero de los algoritmos exactos [20, 21, 78] usa un enfoque basado en una modelización del problema mediante programación lineal (PL) entera y su resolución mediante una estrategia de *branch and cut* que utiliza heurísticas para construir cotas inferiores en los nodos de ramificación. El problema Max-CMO se reduce al de encontrar el máximo conjunto independiente (MIS) en grafos especiales que contienen aproximadamente 10000 vértices para instancias de Max-CMO de aproximadamente 300 residuos.

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 (LP) 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 PL varía entre 1 minuto y dos horas, con lo que este algoritmo sólo es útil en la comparación de las instancias más pequeñas del problema. No obstante, es un algoritmo importante, puesto que fue el primero en garantizar la respuesta óptima, lo que es fundamental, por ejemplo, para establecer comparaciones fiables de la calidad de distintos métodos en términos de optimización.

Más recientemente, en [140, 141], se presentó otra estrategia para resolver de forma óptima el Max-CMO. Esta nueva estrategia no realiza la transformación del Max-CMO a otro tipo de problemas, sino que trata de resolverlo de forma directa, explotando la estructura matemática del problema para desarrollar esquemas eficientes de reducción y de cotas inferiores y superiores.

El algoritmo, de forma similar al primer exacto, emplea una estrategia de ramificación y acotamiento (*branch and bound*) para garantizar que se obtiene una solución óptima, pero como hemos dicho, trabaja sobre el problema original, sin hacer transformaciones a un problema distinto. El algoritmo comienza permitiendo, en el nodo raíz del árbol de búsqueda, todas las parejas posibles de residuos como parte de la solución final, computando cotas inferiores y superiores del valor objetivo de la instancia de Max-CMO que está siendo resuelta. Si las cotas coinciden se tiene una solución óptima con el valor de dicha cota. Sino, se selecciona un par de residuos alineados entre las dos proteínas ($x \in P_1, y \in P_2$) y se crean dos descendientes del nodo, forzando la presencia del par seleccionado en la solución en uno de los nodos y prohibiéndola en el otro.

El proceso de ramificación y acotamiento se repite en estos nuevos nodos y sus descendientes, eliminando aquellos nodos cuya cota superior sea inferior a la mejor solución obtenida hasta el momento. Cada nodo del árbol de búsqueda corresponde a una instancia a resolver del problema Max-CMO, con restricciones adicionales que incluyen los pares seleccionados (que deben estar en la solución), los pares no permitidos (que no se pueden añadir a la solución) y los pares libres que pueden incluirse en la solución si son seleccionados para ello. En cada nodo se aplican cuatro criterios de reducción adicionales que permiten acelerar la búsqueda eliminando pares que se cruzan con los existentes en la solución parcial, pares que se prueba que no son necesarios para una solución óptima y pares de los que existen otros que pueden sustituirlos sin que eso empeore la solución. Para el cálculo de la cota superior el algoritmo usa una relajación que sobreestima el valor objetivo y la cota inferior se calcula encontrando una solución factible, probándose que los esquemas de cota inferior y superior son válidos. Además, al realizar la ramificación, el algoritmo escoge los pares basándose en el potencial máximo para crear ciclos y en el máximo incremento puntual del valor objetivo. Esta elección pretende escoger la pareja que mejor se corresponde, de forma que el problema resultante pasa a ser más sencillo.

En los resultados experimentales, el nuevo algoritmo exacto se ha mostrado más eficiente que el exacto anterior, siendo capaz de resolver un número mayor de instancias en los experimentos realizados. No obstante, aún existen instancias en que el tiempo se dispara y no se puede alcanzar ninguna solución en un tiempo razonable.

Pese a la presencia de los algoritmos exactos comentados, existen aún varias razones que justifican la necesidad de aplicar también algoritmos aproximados, propios del área de la Soft Computing, para la resolución del Max-CMO:

- el problema de maximizar la superposición entre dos mapas de contacto es un problema NP-completo [50, 75], con lo que existirán instancias particulares del problema, es decir, pares concretos de proteínas, en los que los algoritmos exactos no podrán ser capaces de devolver una solución en un tiempo razonable.

- los algoritmos exactos son costosos y difíciles de codificar. Por ejemplo, pueden implicar (como en [20]) el uso de una estrategia de búsqueda local o incluso un algoritmo genético (con su correspondiente establecimiento adecuado de parámetros) para obtener cotas inferiores, o de un algoritmo de programación lineal para obtener cotas superiores. Más aún, si se establece un límite en el tiempo de ejecución, los algoritmos exactos podrían terminar sin proporcionar ningún tipo de solución.

- la disponibilidad de métodos exactos es limitada y sólo el algoritmo presentado en [140] está disponible (http://eudoxus.scs.uiuc.edu/cmos/cmos.html). Sin embargo, existen restricciones tanto en el tamaño de los problemas enviados (no más de 100 residuos) como en el tiempo de procesador que se le concede para la resolución (un máximo de 10 minutos).

- Max-CMO pretende maximizar una relación puramente geométrica entre grafos con lo que un conjunto de soluciones subóptimas podrían también proporcionar descubrimientos en términos del significado biológico del alineamiento.

- debido a errores potenciales en la determinación de las coordenadas en 3D, se puede cuestionar la utilidad de tener soluciones exactas para pares de proteínas que provienen de mapas de contacto que no son totalmente correctos. Como se estableció en [81], los errores experimentales en la determinación de las coordenadas atómicas cartesianas por Cristalografía de Rayos X o NMR varían desde 0.01 a 1.27Å, lo que está cerca del valor de algunos enlaces covalentes.

Es por ello que, como parte del desarrollo de esta tesis, se ha desarrollado un algoritmo heurístico de búsqueda por entornos variables (VNS) para el problema Max-CMO. Como se verá en el capítulo 4, este algoritmo es capaz de obtener resultados cercanos al óptimo usando cantidades limitadas de recursos computacionales y tiempo.

2.2.9. GMax-FCMO: Problema generalizado de la máxima superposición de mapas de contacto difusos

Cuando en lugar de mapas de contacto normales se utilizan mapas de contacto difusos (ver sección 2.2.6), el Max-CMO no puede aplicarse directamente. Para solventarlo hay que recurrir a una generalización, como es el caso del problema generalizado de la máxima superposición de mapas de contacto difusos o GMax-FCMO [112], que, al igual que en el caso del Max-CMO, es un problema NP-completo, como veremos enseguida.

La formulación y resolución del GMax-FCMO como un problema de optimización es, asimismo, casi idéntica al caso del Max-CMO, incluyendo la restricción que evita los cruces entre aristas y siendo el objetivo encontrar la superposición máxima. La diferencia principal estriba en el cálculo de la aportación de cada ciclo de longitud 4 al valor final de la superposición. Mientras que en el modelo Max-CMO, al ser considerados *iguales* todos los tipos de contactos, cada ciclo simplemente suma una unidad al valor objetivo, en el modelo GMax-FCMO, con el uso de mapas de contacto difusos, se contempla la semántica adicional que los tipos de contactos proporcionan y la existencia de grados de pertenencia asociados a cada tipo de contacto. En la figura 2.11 se puede ver un alineamiento entre dos mapas de contacto difusos en los que se distinguen en diferentes colores dos tipos de contacto. Se puede observar que el alineamiento realizado empareja contactos de tipos diferentes en el ciclo ($2 \in P_1, 6 \in P_1, 3 \in P_2, 7 \in P_2$). Es decir, se emparejan dos contactos con un significado diferente, lo cual representa un "error semántico" dado que dichos contactos vienen de funciones de pertenencia diferentes. Dado que estos contactos no deberían ser alineados, en el GMax-FCMO se establece una penalización para este tipo de alineamientos.

En concreto, la contribución de un ciclo $s = (i, j, \sigma(i), \sigma(j))$ al valor de la superposición ($P(s)$) viene dada por la ecuación

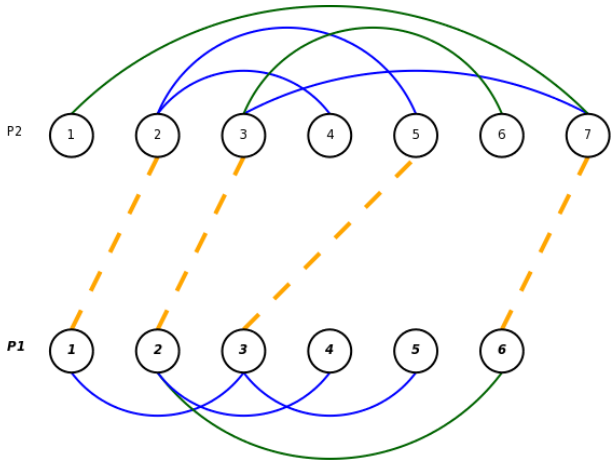


Figura 2.11: Ejemplo de un alineamiento entre dos mapas de contacto difusos.

$$P(s) = (c_{i,j} * c_{\sigma(i),\sigma(j)}) * (t_{i,j} \otimes t_{\sigma(i),\sigma(j)}) \tag{2.12}$$

donde $c_{i,j}$ es el grado de pertenencia asociado al contacto entre los residuos $i \leftrightarrow j$, $t_{i,j}$ es el tipo de dicho contacto y el operador $\otimes$ retorna 1 si sus argumentos son iguales y -1 en caso contrario. De esta forma, se consigue asociar la penalización deseada a los errores semánticos de alinear contactos de tipos diferentes.

Considerando la definición anterior, y que el problema Max-CMO es NP-completo, es fácil ver por qué el GMax-FCMO es también un problema NP-completo. En efecto, el Max-CMO queda englobado por el GMax-FCMO, del que no es más que un caso particular en el que la función de pertenencia es única (un único tipo de contactos) y establece un grado de pertenencia constante de 1 para todas las distancias entre 0 y el umbral máximo.

2.3. Conclusiones

En este capítulo hemos presentado el campo de la Bioinformática, en el que se concentra la mayor parte del trabajo de esta tesis. Hemos visto como se ha producido su aparición en el marco propicio que suponía el éxito del Proyecto Genoma y el gran volumen de datos y la complejidad de los nuevos problemas que se abordan en el campo de la biología, así como la

disponibilidad de financiación con vistas en los grandes beneficios potenciales que puede tener en los campos de la medicina o la biotecnología.

Posteriormente, hemos presentado el área concreta de la Bioinformática estructural en la que se encuadran las herramientas y experimentos desarrollados en esta tesis. Se ha hecho especial hincapié en la descripción de las proteínas, su importancia y numerosas funciones, así como en cómo la estructura determina en buena medida la función, lo que hace que la comparación de estructuras de proteínas tome un papel muy importante en la investigación para fabricar nuevas proteínas o medicamentos.

La comparación de proteínas es a priori un problema que difícilmente puede ser tratado sin amplios conocimientos de Biología. No obstante, el hecho de que la función y la estructura de las proteínas estén relacionadas y la disponibilidad de multitud de datos de posiciones tridimensionales de los átomos de las proteínas abre la vía a la creación de modelos que permitan hacer más accesible la resolución de este problema bioinformático a los investigadores del área de la informática, permitiéndonos hacer aportaciones útiles para los biólogos mediante la creación de herramientas que les permitan realizar parte de su investigación sin tener que pasar forzosamente por el laboratorio, reduciendo tiempo y costes. En concreto, se han presentado dos modelos para modelización de la estructura de las proteínas mediante la transformación de las coordenadas 3D (o las matrices de distancia) en contactos entre los residuos. De esta forma, una proteína queda modelada por un grafo de contactos entre sus residuos y es factible aplicar técnicas computacionales para su resolución. Al tratarse de problemas NP-completos, los algoritmos de tipo exacto no pueden resolver todas las instancias en un tiempo razonable, y se hace recomendable la utilización de técnicas de Soft Computing como son las metaheurísticas. Esta tesis presenta aportaciones en dicho terreno. En concreto, en el capítulo 4 veremos la aplicación de un nuevo algoritmo de búsqueda por entornos variables para la resolución del modelo Max-CMO, y en el capítulo 5 se estudia la comparación de proteínas mediante mapas de contacto difusos tanto a través de la medida universal de similaridad como de una nueva versión del algoritmo de búsqueda por entornos variables.

Capítulo 3

SiGMA: Un esqueleto para Sistemas de Ayuda a la Decisión basados en optimización

En los capítulos anteriores se ha dado una introducción a la Soft Computing y al campo de la Bioinformática, presentando los problemas que va a abordar esta tesis y los modelos que se utilizarán para resolverlos de forma computacional. Al ser estos problemas NP-completos, ya hemos visto que es necesario desarrollar algoritmos heurísticos que puedan dar soluciones cercanas a la óptima en aquellas instancias en las que los algoritmos exactos no son capaces de dar soluciones en tiempo razonable. Además, dado que pueden existir errores como pueden ser los fallos en las mediciones empleadas para obtener los datos de las proteínas (que conforman las instancias de nuestros problemas), la necesidad de obtener la solución óptima es relativa y la utilidad de los algoritmos heurísticos toma mayor relevancia, acentuándose aún más la importancia de su mayor velocidad en alcanzar la solución y aconsejándose la creación de modelizaciones difusas del problema, como la que hemos visto para los mapas de contacto en el capítulo anterior.

Si tomamos también en cuenta el gran volumen de proteínas y, por tanto, de instancias potenciales del problema de comparación de estructuras de proteínas, con lo que ello supone tanto en el manejo de estos datos y sus resultados como en la necesidad de disponer de múltiples algoritmos

heurísticos, ya que ninguno será el mejor para todas las instancias posibles, nos encontramos con una situación que no es realista manejar de forma manual, ejecutando una a una las instancias con cada algoritmo deseado y recopilando y clasificando todos los resultados.

Teniendo en cuenta lo anterior, quedó claro muy pronto, durante el planteamiento de la tesis, que no podíamos llegar a buen puerto sin un Sistema de Ayuda a la Decisión que nos permitiera afrontar la situación anterior de forma sistematizada y lo más automática posible. Tras el estudio del área de los SAD que se hizo en la sección 1.2, se concluyó que el sistema necesario para esta tesis, de nombre SiGMA, y que va a ser presentado a continuación, debía ser un SAD basado en optimización, genérico y extensible de forma dinámica, con la capacidad para proporcionar las siguientes características y funcionalidades:

- Facilitar la incorporación de *resolutores* preexistentes sin tener que reimplementarlos en ningún lenguaje específico, lo cual es necesario porque los usuarios del SAD no estarán en general dispuestos a realizar esta tarea, o podrían carecer del conocimiento y del tiempo necesarios para ello.

- Recolectar y mostrar al usuario cualquier resultado obtenido por los *resolutores* de manera sencilla y sistemática.

- Permitir agregar, modificar y borrar *resolutores* de un modo dinámico, incluyendo la generación dinámica de las interfaces gráficas de usuario (GUI) necesarias para la ejecución de los *resolutores* y el análisis de sus resultados.

- No requerir la recompilación de los *resolutores* o del SAD durante el trabajo normal con el sistema, incluyendo las tareas relacionadas con las características previas.

- Proporcionar una base de datos de *resolutores* que permita reutilizarlos o modificarlos en cualquier momento, para poder olvidarse de los detalles específicos de configuración y parámetros de los *resolutores* y concentrarse en su uso y el análisis de sus resultados.

- Debería estar basado en herramientas de código abierto y lenguajes bien establecidos, de forma que el SAD en sí pueda ser fácilmente extendido o portado a distintas plataformas.

- Debería poder ejecutarse en los principales ordenadores y sistemas operativos, de forma que no se imponga una restricción al entorno de trabajo de un decisor. Así se evitaría el inconveniente de tener *resolutores* que sólo funcionan en una determinada plataforma, al poder llevar el SAD a la misma.

SiGMA es esencialmente un esqueleto (framework) para el desarrollo de SAD basados en optimización a cuya presentación se dedica este capítulo. Así, se hará en primer lugar una descrición inicial de SiGMA, explicando luego las tres partes principales de la herramienta y su modo de uso. Posteriormente se mostrarán dos ejemplos de aplicación de SiGMA para la construcción de sendos prototipos iniciales de SAD para los problemas del p-hub y de la comparación de estructuras de proteínas. Terminará el capítulo con algunas ideas y comentarios sobre la repercusión y los logros de SiGMA.

3.1. Descripción, requisitos y diseño de SiGMA

Un SAD basado en optimización se puede dividir en tres etapas: Formulación, Solución y Análisis. La etapa de Formulación se encarga de la generación de un modelo en una forma que sea aceptable por un *resolutor* del modelo (solver); la etapa de Solución se asocia a la resolución algorítmica del modelo por un *resolutor* apropiado; y la etapa de Análisis engloba el análisis e interpretación de una solución o conjunto de soluciones.

SiGMA es un esqueleto (framework) para la construcción de SAD basados en optimización, que se enfoca especialmente a la etapa de Solución, proporcionando un gestor dinámico para *resolutores* algorítmicos que facilita su manejo, comparación y el análisis de sus resultados, independientemente del problema o modelo a resolver para el que han sido diseñados y del formato de entrada que esperan recibir.

Para alcanzar las metas de un SAD basado en optimización establecidas al principio del capítulo, SiGMA proporciona al usuario la capacidad de añadir, modificar, ejecutar y ver los resultados de cualquier *resolutor*, con una interfaz gráfica de usuario sencilla y sin tener que escribir una sola línea de código. Una vez que un *resolutor* ha sido especificado completamente por el usuario, SiGMA es capaz de generar las interfaces de tipo formulario apropiadas para ejecutarlo y recolectar su salida, haciéndola asimismo fácilmente disponible para su análisis. En lo que resta de este capítulo, un

resolutor se estará refiriendo a un algoritmo, o, más concretamente, a una implementación de un algoritmo, y ambos términos serán usados de forma intercambiable.

Muchos escenarios pueden verse muy beneficiados por el uso de un sistema como SiGMA. Dos ejemplos son:

1. Un decisor que quiere incorporar un buen número de *resolutores* para el problema en el que trabaja. Por ejemplo, un corredor de bolsa durante su trabajo en el mercado bursátil, podría beneficiarse de cualquier nuevo algoritmo que analice los movimientos de las acciones y sugiera operaciones de compra o venta. De esta forma, podría analizar y agregar/combinar los resultados de los diferentes *resolutores* y usar su propio conocimiento experto para tomar la decisión final de qué acciones llevar a cabo. SiGMA encajaría perfectamente en este escenario al facilitar la rápida inclusión y ejecución de nuevos *resolutores*.

2. Cuando un investigador desarrolla un nuevo algoritmo (*resolutor* para un problema específico), quiere tener la posibilidad de compararlo con todos los algoritmos para el mismo problema, o, al menos, los mejores disponibles en la literatura. Esto permitiría al investigador conocer cuáles son los puntos débiles y fuertes de su nuevo algoritmo, y las ventajas y desventajas de usar un algoritmo frente a otro. SiGMA se adecúa especialmente bien a este escenario, permitiendo fácilmente añadir y manejar en el sistema todos los *resolutores* que necesitan ser comparados. Y este escenario se da, por ejemplo, en el problema de comparación de estructuras de proteínas que aborda la tesis.

Los *resolutores* accesibles para un usuario concreto, pueden estar disponibles como código fuente completo o solamente como ficheros binarios ejecutables. Aunque existe la posibilidad de modificar el código de los algoritmos cuando está disponible, los ficheros binarios ejecutables no pueden modificarse, con lo cual, si se exige algún requisito en este punto, se imposibilitaría su uso. Además, aunque el código fuente de un algoritmo esté disponible, dicho código puede ser muy complejo o difícil de modificar. La modificación podría requerir destreza en el manejo de nuevos lenguajes que el investigador no tiene por qué tener o cuyo aprendizaje no es rentable. Tomando todo esto en consideración, está claro que la integración de *resolutores* arbitrarios en un SAD no es una tarea sencilla, e, idealmente, el usuario que trabaja

con el SAD no debería tener que conocer nada de su funcionamiento interno para ser capaz de añadir un nuevo *resolutor* al sistema.

Por tanto, la primera decisión que se tomó para hacer de SiGMA un esqueleto para SAD basados en optimización lo más genérico posible, fue imponer que trabajara con ficheros binarios ejecutables directamente, eliminando de esta forma la necesidad de realizar modificaciones en los ficheros de código fuente. De manera más precisa, SiGMA trabajará con cualquier fichero binario ejecutable que reciba sus parámetros mediante la línea de comandos y que produzca su salida bien mediante ficheros o bien mediante su salida estándar. De esta forma, el tiempo y esfuerzo necesario para añadir un nuevo *resolutor* a SiGMA será simplemente el tiempo necesario para especificar dónde está el ejecutable, qué parámetros recibe y qué tipos tienen dichos parámetros, más los metadatos opcionales que se crea necesario, como puede ser una descripción del *resolutor*. SiGMA proporcionará las interfaces gráficas de usuario apropiadas para el cumplimiento de estas tareas.

Como no hay ningún requerimiento o restricción en el formato de la salida estándar de un *resolutor* o en los ficheros que produce, no es posible que se proporcione una etapa de Análisis compleja en SiGMA. Por este motivo, SiGMA sólo proporcionará un análisis básico de los resultados, pudiéndose añadir análisis de resultados más avanzados, cuando se trabaja con un problema específico, mediante la adición de dicha funcionalidad extra sobre la base del esqueleto que SiGMA proporciona.

3.1.1. Detalles técnicos

Para cumplir todos los requerimientos anteriores, SiGMA ha sido desarrollado utilizando tecnologías Java [59]. Java es un lenguaje bien conocido y ampliamente aceptado que desarrolla la empresa Sun Microsystems. El código fuente en Java es compilado a una representación binaria que puede ejecutarse en una máquina virtual de Java (Java Virtual Machine o JVM). Como la JVM oculta toda la arquitectura de la máquina real específica sobre la que se está ejecutando, un programa Java puede ejecutarse sin modificaciones en cualquier arquitectura y sistema operativo para el que haya disponible una máquina virtual apropiada. Gracias a su gran difusión y a que el código fuente tanto de la JVM en sí, como del compilador de Java y las herramientas asociadas están disponibles como código fuente abierto, ya existen gran cantidad de máquinas virtuales de Java y es posible modificar

y compilar el código para otras arquitecturas si fuera necesario. Particularmente existen JVMs estables y funcionando perfectamente para todas las principales máquinas y sistemas opreativos, es decir: Microsoft Windows, la mayoría de versiones de Unix y Linux, y Apple Mac OS.

No obstante, aunque la máquina virtual de Java es una buena abstracción del hardware y el sistema operativo subyacente, no todas las tareas se realizan de forma exactamente igual en todos los sistemas operativos. Por ejemplo, la ejecución efectiva de programas y la apertura de un fichero con su programa asociado son tareas distintas, en algunos aspectos, según la plataforma donde se esté trabajando. Afortunadamente, la cantidad de cambios necesarios para adaptar estas tareas a un sistema operativo diferente no es demasiado grande, y SiGMA ha sido preparado para trabajar correctamente en algunas de las últimas versiones de los tres principales sistemas operativos. Se han realizado pruebas para comprobar que el funcionamiento de SiGMA es correcto en Microsoft Windows XP, Kubuntu Linux 7.04 y Apple Mac OS X Tiger.

En cuanto al almacenamiento de los datos manipulados por SiGMA, la decisión tomada fue la de emplear el sistema de ficheros para almacenar tanto los ficheros de definición de los *resolutores* como sus resultados. Los ficheros de definición van a ser almacenados mediante ficheros XML usando las clases XMLEncoder y XMLDecoder del paquete java.beans, y la base de datos completa de *resolutores* será simplemente la serialización de una clase de tipo lista que los contenga. Esta aproximación es extremadamente simple y poderosa, al permitir, por ejemplo, compartir fácilmente las definiciones de *resolutores* entre distintas instancias de SiGMA mediante la simple copia de sus ficheros de definición en XML al lugar apropiado. Los resultados de los *resolutores* van a ser también ficheros (incluyendo un fichero que recogerá la salida estándar) que se almacenarán de forma directa en el sistema de ficheros de la máquina en la que se ejecuta SiGMA.

Todos los ficheros generados durante la ejecución de SiGMA se crean y almacenan en el subdirectorio `SiGMA` del directorio personal del usuario. Este directorio depende del sistema operativo, así que si *sigmauser* es el nombre de la cuenta del usuario que trabaja con SiGMA, el directorio personal asociado será: `C:\Documents\and\settings\sigmauser\`, si el usuario trabaja con Microsoft Windows XP; `/home/sigmauser/`, si el usuario trabaja con casi cualquier distribución de Unix / Linux (en su configuración por defecto);

o `/Users/sigmauser/`, si el usuario utiliza Mac OS X. SiGMA simplemente crea un subdiretorio llamado `SiGMA` en la ruta apropiada para el directorio personal y lo utiliza para todos sus datos. Esto significa que se almacenan en dicho subdirectorio dos cosas principalmente: los ficheros de definición de los *resolutores*, y los resultados de las ejecuciones de los mismos. Todos los *resolutores* se almacenan en el subdirectorio `solvers`, y los resultados se almacenan en subcarpetas dentro del subdirectorio `results`.

3.2. Trabajando con SiGMA

En la figura 3.1 se muestra la pantalla inicial de SiGMA, que se compone de una barra de menús, y un panel con tres pestañas: 1) la pestaña *Solver Administration*, que permite al usuario añadir, editar y borrar *resolutores* (algoritmos); 2) la pestaña *Solver Execution*, donde se establecen los valores de los parámetros de los *resolutores* y se permite su ejecución; y 3) la pestaña *Result analysis*, donde pueden verse los resultados recolectados tras las distintas ejecuciones de los *resolutores*.

Los resultados en SiGMA se agrupan en sesiones y tareas. Una sesión es simplemente un nombre común para un grupo de tareas, mientras que una tarea se asocia a la ejecución de un *resolutor* específico. SiGMA crea subcarpetas para cada sesión dentro de su subdirectorio `results` y, asimismo, cada tarea se almacena en su propia subcarpeta dentro de la carpeta de sesión asociada (se incluyen más detalles sobre los nombres dados a estas carpetas y los contenidos de cada directorio de tarea al describir la ejecución de los *resolutores* en la sección 3.2.2).

El árbol de directorios que se genera es muy apropiado para realizar copias de seguridad de los datos de los *resolutores* y de los experimentos, porque basta con preservar este único árbol de directorios en la copia. Los datos de *resolutores* y sus resultados pueden también ser fácilmente compartidos entre usuarios de diferentes instalaciones de SiGMA mediante la simple copia de los ficheros y directorios a la instalación de destino, manteniendo la misma estructura de directorios. SiGMA está preparado para comprobar si existen nuevos resolutores al comienzo de la ejecución y para añadirlos automáticamente a la base de datos de *resolutores*. Además, una vez que se tienen resultados de cualquier *resolutor*, estos resultados pueden ser copiados para ser vistos en cualquier otra instalación de SiGMA, incluso

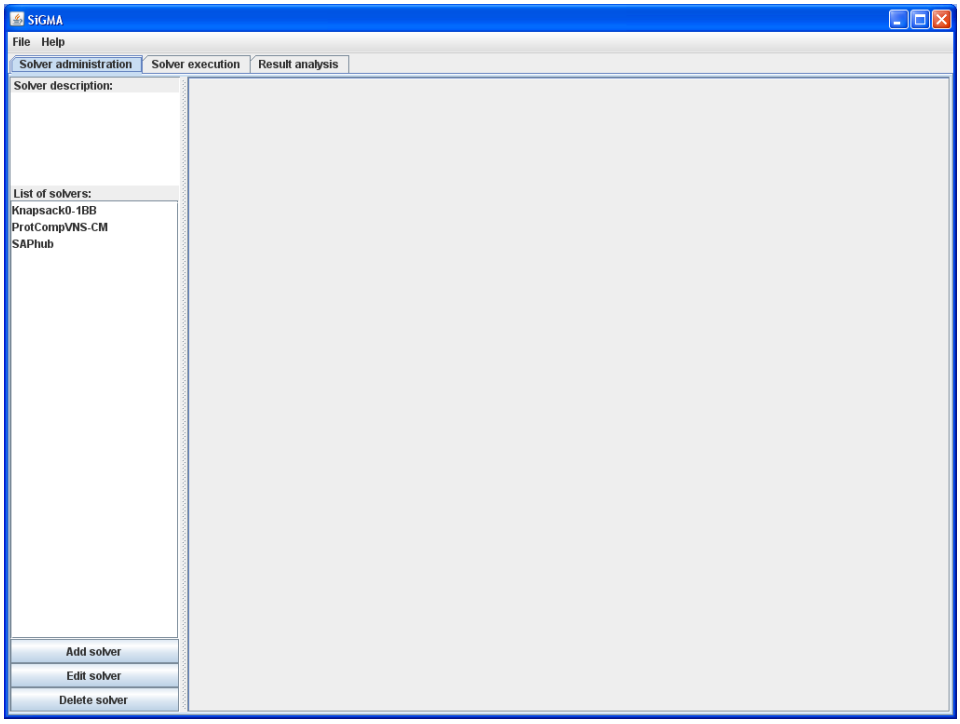


Figura 3.1: Ventana principal de SiGMA.

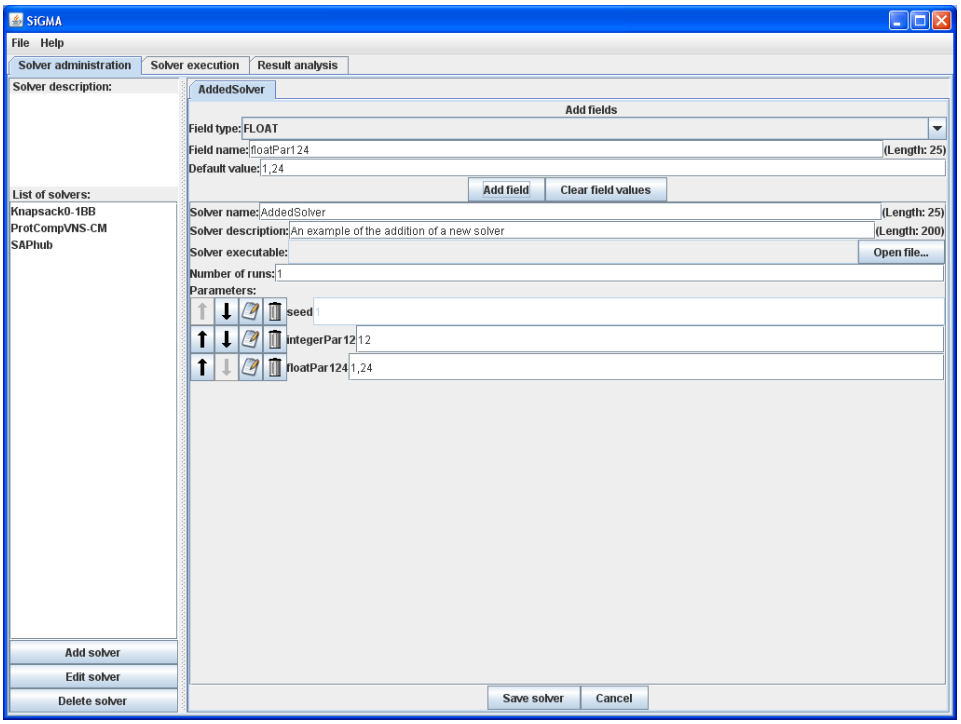


Figura 3.2: Pestaña Solver Administration de SiGMA: Añadiendo un *resolutor*.

si la nueva instalación no dispone de los *resolutores* que generaron dichos resultados, ya que no son necesarios en la fase de análisis.

Las siguientes subsecciones describen cada pestaña de SiGMA con mayor detalle.

3.2.1. Administración de resolutores

La pestaña *Solver Administration* sirve para que el usuario añada, edite y borre *resolutores*. Esta pestaña se divide verticalmente en dos partes. La parte izquierda contiene una descripción del *resolutor*, una lista de *resolutores* y tres botones (Add, Edit y Delete solver).

Un clic en el botón "Add solver" carga una nueva pestaña en la parte derecha de la pestaña *Solver Administration* que permite agregar un nuevo *resolutor*, como se muestra en la figura 3.2.

Todos los *resolutores* necesitan un nombre, una descripción y el ejecutable del *resolutor*, que se especifican en los tres primeros campos bajo el

botón “Add field”. A continuación deben especificarse los parámetros del algoritmo, para lo cual hay una lista desplegable que permite seleccionar un tipo de campo para el parámetro a añadir. Bajo esta lista aparece una serie de campos, dependientes del tipo de parámetro, que permiten especificar las características del mismo, y el botón “Add field” que permite al usuario añadir el campo al *resolutor*. SiGMA realiza un chequeo de los parámetros para asegurarse de que cada parámetro tiene valores correctos para su tipo, descartando o corrigiendo las entradas erróneas, y mostrando los valores en el formato correcto para la configuración regional (país) del usuario.

En la figura 3.2 se muestra un ejemplo en el que se han añadido ya tres campos al *resolutor*, y se muestran bajo la etiqueta “Parameters”: una semilla aleatoria (seed), un parámetro “integerPar12” de tipo entero, con un valor por defecto de 12, y un parámetro “floatfield124” de tipo flotante y valor por defecto 1,24. Los tipos de parámetros disponibles son: *INTEGER* y *FLOAT*, que se definen mediante un nombre y un valor por defecto; *STRING*, que se define por un nombre, un valor por defecto y una longitud máxima; y *RANDOM_INTEGER_SEED*, *FILE* y *FILE_OF_VALUES*, que se definen simplemente con un nombre. Una vez que se obtienen los datos requeridos, el nuevo *resolutor* puede añadirse a la base de datos de *resolutores* pulsando el botón “Save solver”. Es importante notar que mientras un parámetro de tipo *FILE* es simplemente un fichero que el algoritmo recibe en sus parámetros como parte de su entrada, el tipo *FILE_OF_VALUES* tiene un significado diferente. En concreto, en lugar de usarse la ruta del fichero como un parámetro del algoritmo, SiGMA lee el fichero línea a línea cuando se va a ejecutar el algoritmo, y por cada valor (línea) que se lee realiza una ejecución con el valor leído como parámetro. Se mostrará un ejemplo práctico del funcionamiento de los campos de tipo *FILE_OF_VALUES* al final de la siguiente sección, donde se va a explicar la ejecución de resolutores en SiGMA.

3.2.2. Ejecución de resolutores

La pestaña *Solver Execution* permite al usuario ejecutar los *resolutores* de manera inmediata o incluyéndolos en una cola de tareas a ser procesadas posteriormente.

La pestaña se divide en dos partes. La parte izquierda contiene una lista de los *resolutores* disponibles, un panel de descripción del *resolutor*

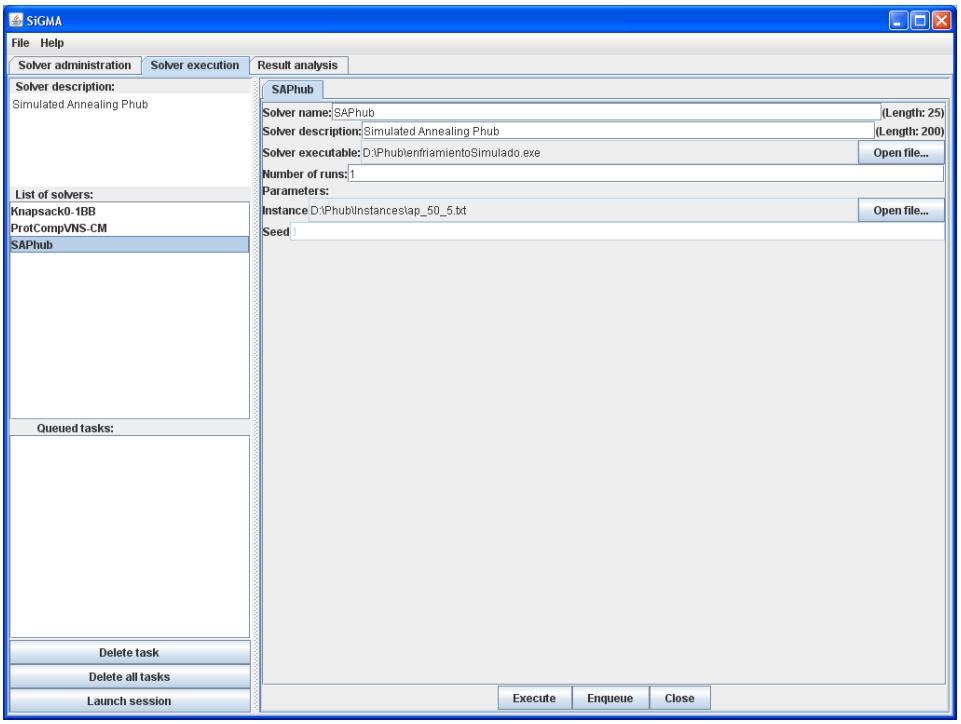


Figura 3.3: Pestaña Solver Execution de SiGMA: Ejecutando un algoritmo.

(correspondiente al *resolutor* actualmente seleccionado), una lista de tareas en cola y tres botones para manipular la lista de tareas: "Delete task", "Delete all tasks" y "Launch session".

Para ejecutar un *resolutor*, hay que abrirlo desde la lista de *resolutores* (para ello basta con hacer doble clic en su nombre) y una nueva pestaña **dinámicamente generada** se abrirá en la parte derecha de la pestaña *Solver Execution*. En esta nueva pestaña tipo formulario, se deben establecer los parámetros del *resolutor* y el número de repeticiones que se desea que se hagan para cada ejecución. A continuación, se debe presionar el botón "Execute" para ejecutar el *resolutor* de manera inmediata o el botón "Enqueue" para añadirlo a la cola de tareas a ejecutar más tarde. La figura 3.3 muestra un ejemplo de la ejecución de un *resolutor* para el problema del p-hub.

Como se ha dicho, un algoritmo puede ser ejecutado "inmediatamente" (presionando el botón "Execute") o puede ponerse en cola para una ejecución posterior (mediante el botón "Enqueue"). En este último caso, la

lista de tareas en cola puede ser manipulada para borrar la tarea seleccionada ("Delete task"), borrar todas las tareas de una sola vez ("Delete all tasks") o lanzar una nueva sesión con todas las tareas almacenadas en la cola ("Launch session").

Una vez que se pide la ejecución de un *resolutor* o una serie de tareas almacenadas en cola, SiGMA invoca los ejecutables apropiados para los *resolutores* con los parámetros dados, realizando dicha operación en una serie de directorios de salida generados para tal fin. Para cada tarea, la ruta de este directorio de salida (relativa al directorio principal de datos de SiGMA) tiene la forma `results/session-dir/task-dir`. Los nombres de los directorios de sesión (*session*) se crean siguiendo el patrón *session-FULLDATE*, donde FULLDATE es la fecha actual en el momento de su creación. Los nombres de los directorios de tarea (*task-dir*) siguen, en cambio, el patrón *SOLVERNAME-SHORTDATE*, donde SOLVERNAME es el nombre del *resolutor* asociado con la tarea y SHORTDATE es la fecha en que la tarea es ejecutada. Tanto FULLDATE como SHORTDATE son representaciones numéricas de la fecha en el formato MesDíaHoraMinutoSegundo, con la única diferencia de que FULLDATE incluye además el año al comienzo. Como una tarea puede también tener múltiples repeticiones asociadas (runs), a la vez que más de una configuración de los valores de los parámetros (cuando se usan parámetros de tipo *FILE_OF_VALUES*), cada tarea puede representar un conjunto de ejecuciones individuales. Estas ejecuciones individuales se almacenan en subcarpetas de la carpeta de la tarea a la que pertenencen, con un nombre que sigue el patrón *ParsN-runM*. Como se ha dicho, cuando se utiliza un parámetro de tipo *FILE_OF_VALUES*, el *resolutor* se ejecuta con más de una combinación de parámetros, cada una de estas combinaciones distintas lleva asociado un número, y es este número el que se coloca como valor de N en la subcarpeta. Para la ejecución con cada combinación de parámetros se pueden realizar varias repeticiones, el valor de M en el nombre de la subcarpeta será el número de la repetición a la que corresponde cada ejecución individual.

En la tabla 3.1 se muestran todas las ejecuciones individuales que se generarían al ejecutar un resolutor que tiene como parámetros una semilla aleatoria, un valor entero y dos ficheros de tipo *FILE_OF_VALUES*, pidiendo además la realización de dos repeticiones para cada ejecución. Puede verse cómo el contenido de los ficheros afecta a las configuraciones de parámetros

	Valores de los parámetros				
Ejecución	**SemAl**	**ent**	**fv1**	**fv2**	**Repetición**
1	398246309	17	1	a	1
2	398246309	17	1	b	1
3	398246309	17	2	a	1
4	398246309	17	2	b	1
5	398246309	17	3	a	1
6	398246309	17	3	b	1
7	1815552449	17	1	a	2
8	1815552449	17	1	b	2
9	1815552449	17	2	a	2
10	1815552449	17	2	b	2
11	1815552449	17	3	a	2
12	1815552449	17	3	b	2

Tabla 3.1: Ejemplo de los valores de los parámetros durante la ejecución de un resolutor. Se asume que se ha lanzado el resolutor pidiendo 2 repeticiones y que como parámetros se tiene una semilla aleatoria (**SemAl**), un valor de tipo entero (**ent**) y dos parámetros de tipo *FILE_OF_VALUES* (**fv1** y **fv2**). El contenido de **fv1** se asume que son 3 líneas con los valores "1", "2" y "3", mientras que **fv2** contiene dos líneas con los valores "a" y "b". En la tabla se muestran todas las ejecuciones individuales producidas para esta configuración.

utilizadas, al usarse todas las combinaciones de parámetros correspondientes a todos los valores presentes en las líneas de ambos ficheros.

3.2.3. Análisis de resultados

La pestaña *Result Analysis* es el lugar donde se muestran los resultados de las ejecuciones de los *resolutores*. Esta pestaña también se encuentra dividida en dos zonas verticales. La parte izquierda muestra un árbol completo del directorio `results` de SiGMA, con todas las *sesiones*, *tareas* y ejecuciones individuales.

Cuando un usuario hace doble clic en una ejecución individual, la parte derecha de la ventana carga una nueva pestaña con una lista de los ficheros de resultado que generó el *resolutor* y muestra la salida estándar que produjo,

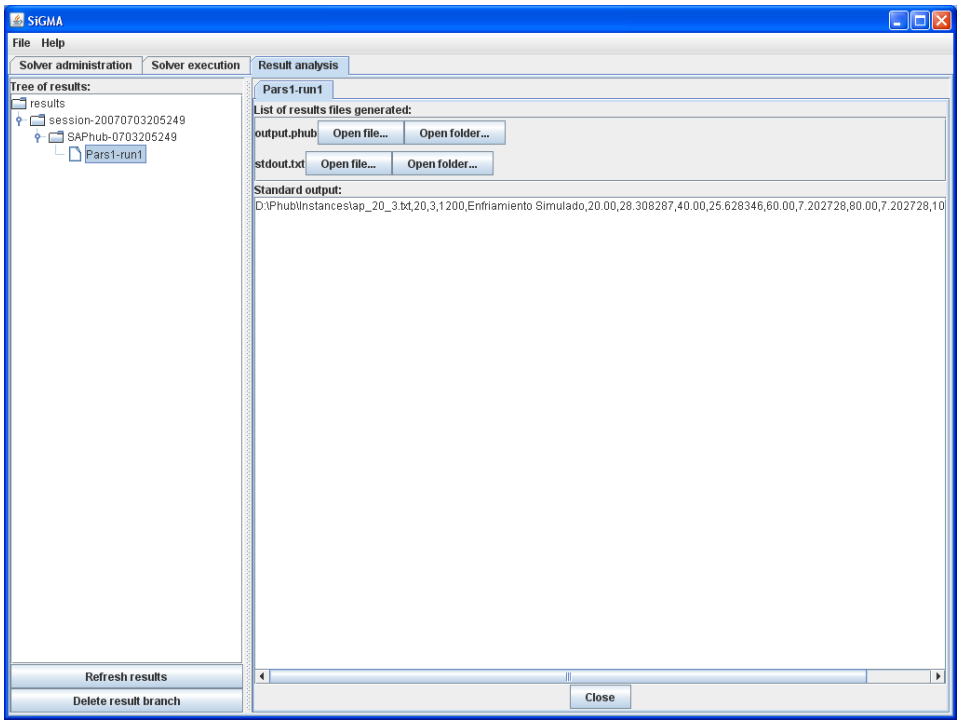


Figura 3.4: Pestaña Result Analysis de SiGMA: Visualización de los resultados de una ejecución.

que además también se almacena en un fichero de texto. Todos los ficheros de resultados van seguidos de dos botones: "Open file..." y "Open folder...", el primero de los cuales abre el fichero con la aplicación asociada al mismo, según su extensión, mientras que el segundo abre la carpeta que contiene dicho fichero. La figura 3.4 muestra un ejemplo de visualización de resultados en SiGMA para un *resolutor* concreto.

También es posible actualizar el árbol de resultados pulsando el botón "Refresh results" o borrar cualquier porción de los resultados mediante la selección de la rama con los resultados a eliminar y la pulsación posterior del botón "Delete results branch".

3.3. Ejemplos de aplicación

Cuando se tiene conocimiento específico del problema que se está resolviendo o de la salida que los *resolutores* producen, suele ser deseable

disponer de un manejo, análisis o visualización de resultados más complejo. El diseño de SiGMA permite la construcción de esta clase de SAD más completo. Ajustar el esqueleto a un problema específico se puede conseguir simplemente extendiendo las clases Java de SiGMA para añadir módulos con la funcionalidad avanzada que es específica del problema o de sus soluciones, a la vez que se reutiliza toda la restante funcionalidad necesaria de la implementación genérica.

Esta sección muestra dos aplicaciones del esqueleto SiGMA a la construcción de SAD más complejos para dos problemas de dominios diferentes: un problema de localización de recursos (el p-hub) y el problema de comparación de estructura de proteínas, que es parte fundamental del contenido de esta tesis. Para mostrar la generalidad del sistema desarrollado, se presenta primero, en la sección 3.3.1, el SAD SiGMAPhub para el problema del p-hub. A continuación, en la sección 3.3.2 se presenta el SAD SiGMAProt, que se utilizó para realizar los experimentos de comparación de proteínas que se describen en los capítulos 4 y 5 de esta tesis.

3.3.1. Problema de localización de facilidades: el modelo del p-hub

El problema del p-hub consiste en seleccionar p facilidades y asignar clientes a dichas facilidades de manera que se minimice la máxima distancia entre un cliente y su facilidad. Es un problema de localización de recursos que aparece en el diseño de redes de telecomunicación, redes de aerolíneas de pasajeros y redes de reparto postal [37]. Se trata de problemas en donde el tráfico entre nodos de una red debe ser conducido a través de una serie de nodos, designados como medianas (hubs). Las medianas, que están completamente interconectadas, facilitan el flujo de tráfico entre los nodos, mientras que los nodos que no son seleccionados como medianas se conectan a la mediana más apropiada. Las medianas sirven, por tanto, de puntos de traspaso o de conmutación para los flujos entre los nodos que no son medianas. Se asocian flujos no negativos para cada par de nodos origen-destino (O-D) de la red. Entre cualquier par de nodos hay un "coste" de envío del flujo entre dichos nodos, basado en factores como el tiempo, el dinero y/o la distancia. Estos flujos son generalmente encaminados a través de una o dos medianas (como máximo). La red resultante tiene una estructura de medianas y radios (las medianas y las conexiones a los nodos no medianas)

y el problema es encontrar las localizaciones óptimas de las medianas y las mejores asociaciones de los nodos no medianas a las medianas (los radios de la estructura). Se pueden presentar restricciones adicionales como, por ejemplo, una capacidad máxima de flujo entre un par de nodos dados, que condicionaría las elecciones a realizar.

Esta sección se centra en la descripción de un ejemplo de SAD basado en optimización para el problema de localización del p-hub mediano sin capacidades [19, 37, 106] sobre el esqueleto que SiGMA proporciona, y haciendo uso de diferentes versiones de algoritmos como el recocido simulado, la búsqueda tabú o la búsqueda por entornos variables. Este sistema ha sido bautizado como SiGMAPhub y, además de la funcionalidad genérica que SiGMA provee, la meta en su construcción fue mejorar sus capacidades en la etapa de análisis mediante la inclusión de una representación gráfica de los resultados de los *resolutores*, que muestra los hubs seleccionados y cómo los restantes nodos son asignados a ellos. Esta salida gráfica se añadió haciendo que las pestañas estándar de visualización de resultados en la pestaña *Result Analysis* de SiGMA se dividiera en un panel con dos nuevas pestañas. La primera de estas pestañas simplemente contiene la salida estándar de la implementación genérica de SiGMA, mientras que la segunda pestaña contiene la representación gráfica de la solución. Se muestra una captura del sistema en la figura 3.5.

Los cambios necesarios en SiGMA para añadir esta capacidad adicional de salida gráfica son muy pequeños gracias al diseño del esqueleto. De hecho, sólo se necesitaron 4 pasos:

1. Crear una nueva ventana SiGMAPhub como extensión (herencia) de la ventana SiGMA original, mostrando SiGMAPhub en lugar de SiGMA en donde fuera necesario.

2. Renombrar la última parte del directorio de trabajo de la aplicación para que fuera SiGMAPhub en lugar de SiGMA y no colisionaran los resultados del sistema genérico y el específico para el p-hub.

3. Crear un visor para mostrar los resultados al problema p-hub que daban los *resolutores* utilizados (con la ayuda de la librería de código abierto JFreeChart [44]).

4. Finalmente, se preparó la pestaña de análisis de resultados para que,

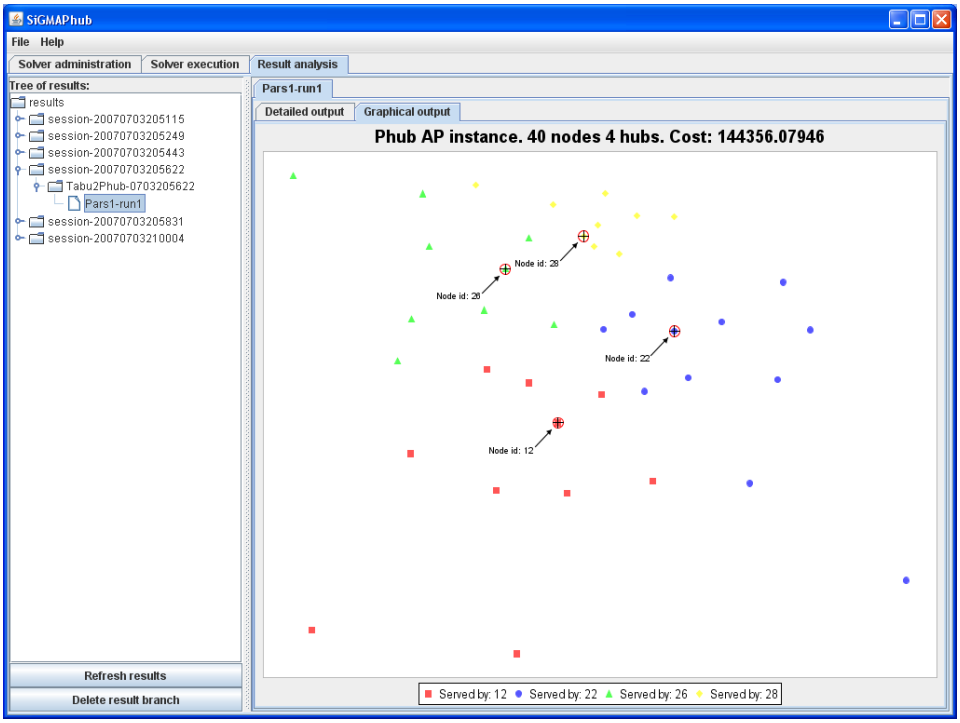


Figura 3.5: Ejemplo de salida gráfica en SiGMAPhub.

al abrir mediante clic una entrada del árbol de resultados, además de mostrar la salida estándar de SiGMA, se incluyese una nueva pestaña que cargara el fichero .phub correspondiente al directorio de salida asociado a la entrada en el visor creado en el paso anterior.

De la misma manera que se ha procedido para crear este SAD para el p-hub, se pueden crear SAD para otro tipo de problemas, extendiendo la funcionalidad deseada del esqueleto de SiGMA para satisfacer los requisitos adicionales que se considere necesario incorporar.

3.3.2. Problema de comparación de estructuras de proteínas

Dado que uno de los focos principales de esta tesis es el problema de comparación de proteínas, el uso natural de SiGMA para ayudar en el desarrollo de la misma y de la experimentación con diferentes algoritmos de comparación de proteínas ha sido realizado mediante una extensión de SiGMA para crear un SAD basado en optimización para la comparación de estructuras de proteínas con una mejor visualización de los resultados, que recibe el nombre de SiGMAProt.

En este problema la mayoría de los algoritmos (*resolutores*) existentes proporcionan una salida que incluye no sólo una medida de similaridad o distancia entre dos proteínas dadas sino también un emparejamiento o alineamiento de residuos de ambas proteínas. Dado un alineamiento de este tipo se puede realizar una superposición de ambas proteínas de manera que se minimice la desviación cuadrática media (root mean square deviation o RMSD) de los residuos alineados. Es decir, se puede realizar una rotación y traslación de una proteína sobre la otra de forma que el valor de RMSD sea mínimo. Esta superposición hace que se hagan coincidir en el espacio las zonas similares de ambas proteínas detectadas por el *resolutor*, y una visualización gráfica de las proteínas superpuestas deja ver claramente la similitud existente.

El proceso de añadir esta funcionalidad fue similar al proceso ya explicado para el caso de SiGMAPhub (sección 3.3.1), con la diferencia de que el visor empleado en este caso ha sido Jmol [71], un excelente visor de código abierto tanto por su calidad gráfica como por su velocidad. En la figura 3.6 se muestra un ejemplo de SiGMAProt en funcionamiento, con la visualización de un par de proteínas superpuestas.

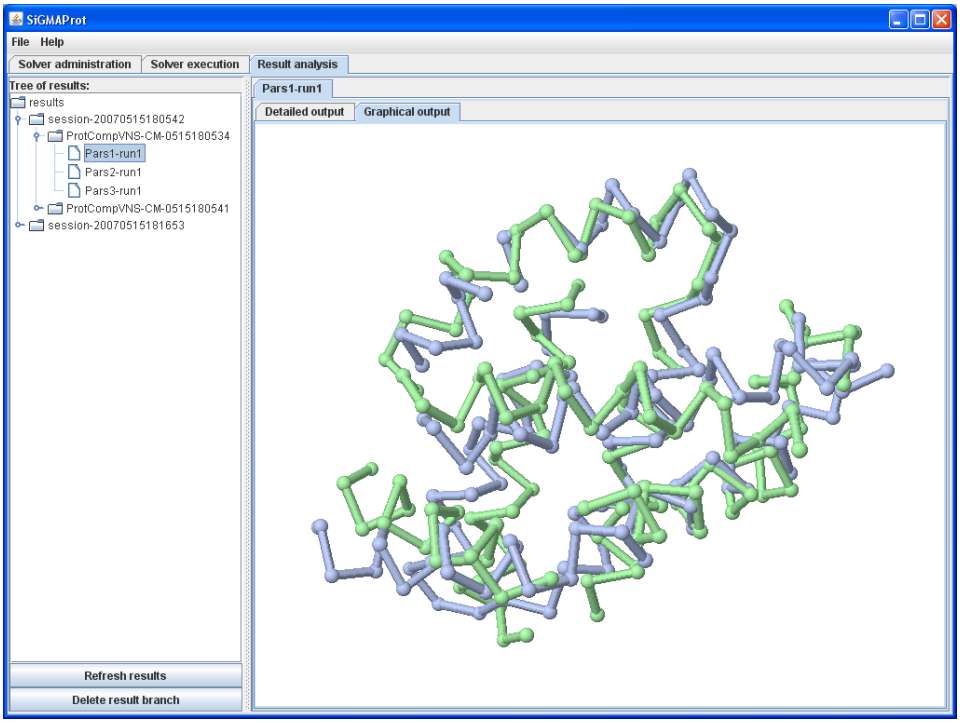


Figura 3.6: Salida gráfica en SiGMAProt (usando Jmol).

Es importante hacer hincapié en que este SAD ha sido clave en el desarrollo de la tesis, al servirnos para realizar la experimentación con los diferentes conjuntos de datos y las diferentes versiones de nuestros algoritmos, siendo la persecución de este objetivo lo que condujo al desarrollo del sistema SiGMA al completo. Una de las ventajas más importantes de SiGMA, de cara al problema de comparación de estructuras de proteínas, es que suele ser necesario realizar una comparación de todos los pares posibles de un conjunto de proteínas. Conseguir esto en SiGMA es tan sencillo como usar parámetros de tipo *FILE_OF_VALUES* para los parámetros donde los *resolutores* esperan recibir la ruta de cada una de las dos proteínas y luego usar como valor de dichos parámetros un fichero conteniendo los caminos de todos los ficheros de las proteínas del dataset (el mismo fichero con los caminos en ambos parámetros referidos a proteínas). De esta forma SiGMA realiza automáticamente la comparación de todas las combinaciones posibles de pares de proteínas. Además, los métodos de comparación de estructuras de proteínas, como pueden ser DaliLite [90], FANS [112] o MatAlign [5] son todos métodos que funcionan mediante la invocación en la línea de comandos, haciendo que sean directamente utilizables por SiGMAProt, sin importar el hecho de que están escritos en lenguajes tan diferentes como Fortran y C++.

3.4. Conclusiones

En este capítulo hemos presentado SiGMA, un esqueleto para el desarrollo de SAD basados en optimización que se enfoca al manejo y ejecución de modo dinámico de los *resolutores* y al análisis de sus resultados. Aunque SiGMA fue diseñado teniendo en mente la experimentación específica necesaria para la tesis, se ha mostrado cómo, en su versión genérica (con la funcionalidad básica del esqueleto), puede cumplir toda la funcionalidad deseada para esta clase de sistemas, tal y como se detalló al principio del capítulo. De esta forma, SiGMA es una herramienta valiosa para el trabajo con cualquier tipo de *resolutores* (algoritmos) de modo independiente de los problemas/modelos que resuelven. Se ha mostrado también cómo se puede usar SiGMA para construir SAD basados en optimización más complejos, que proporcionan funcionalidades más avanzadas, cuando se tiene conocimiento específico del problema, dando dos ejemplos: SiGMAPhub, para el

problema del p-hub mediano, y SiGMAProt, para la comparación de estructuras de proteínas.

Se ha conseguido contruir una herramienta multiplataforma y multilenguaje (en lo que a los *resolutores* se refiere) que genera dinámicamente la interfaz de usuario para la manipulación de los algoritmos mediante el uso de componentes Java Swing [33] y cuyo único requisito es tener una máquina virtual de Java instalada. Un usuario que quiera usar SiGMA sólo necesita ejecutar el fichero `.jar` en el que se distribuye SiGMA y configurar los ejecutables apropiados de los algoritmos que quiere usar como *resolutores*, haciendo uso de la interfaz de SiGMA para este propósito. Una vez que los *resolutores* han sido incorporados, pueden ejecutarse fácilmente y sus resultados se pueden inspeccionar con facilidad.

Las ventajas de usar SiGMA como herramienta para el manejo de algoritmos resultan claras desde varios puntos de vista. Uno de ellos es la ayuda que representa para mantener diferentes algoritmos para cada problema en una base de datos de *resolutores*, de manera que no es necesario recordar los detalles específicos de su ejecución o el orden de sus parámetros, ya que son convenientemente manejados por SiGMA. Otra ventaja clara es la posibilidad de usar una misma aplicación para la ejecución y el análisis de resultados, lo que reduce el tiempo necesario para hacer nuevos experimentos y chequear los cambios de rendimiento de los algoritmos que estamos desarrollando al mismo tiempo que dichos cambios se producen. Además, SiGMA (y de forma similar cualquier SAD basado en SiGMA, como SiGMAPhub y SiGMAProt) no se ve afectado por los cambios en los ejecutables de los algoritmos, con lo que pueden cambiarse y recompilarse tanto como se necesite sin ningún efecto ni necesidad de gestión adicional para reusarlos con las interfaces dinámicas que SiGMA autogenera. Solamente si se modifica la cantidad, el orden o el tipo de los parámetros de un algoritmo, lo cual es poco frecuente, será necesario modificar también la definición del mismo en SiGMA para reflejar dichos cambios. Pero, incluso en esta situación, SiGMA proporciona una interfaz muy sencilla y rápida para la edición de los *resolutores*, haciendo que sea una tarea trivial.

Por otro lado, en la investigación realizada durante el desarrollo de esta tesis, en especial en lo referente a la comparación de estructuras de proteínas y los experimentos de los capítulos 4 y 5, se hizo un uso extensivo de SiGMAProt como método para facilitar la ejecución de los *resolutores*

sobre grandes conjuntos de datos, y para recuperar fácilmente la salida de cara a la realización de otro tipo de análisis más complejos con paquetes estadísticos. Cabe hacer notar pues, que, aunque no se indique de forma específica en todos y cada uno de los experimentos que se presentarán, su realización manual no habría sido posible en el tiempo disponible para esta tesis. En particular, ha sido esencial la capacidad de SiGMAProt para poder ejecutar comparaciones de múltiples pares de proteínas en una única sesión de ejecución gracias a los campos de tipo *FILE_OF_VALUES*.

SiGMA ha sido presentado en los congresos MIC 2005 [58] y MAEB 2005 [54] en forma de versiones preliminares y está enviado también, en su versión actual, a la revista Expert Systems With Applications (ESWA) [57]. Asimismo, SiGMA está disponible para los interesados a través de petición por correo electrónico.

Capítulo 4

Resolución del modelo Max-CMO de comparación de estructuras de proteínas

En los capítulos previos hemos presentado las áreas de Soft Computing, Sistemas de Ayuda a la Decisión y Bioinformática; y hemos mostrado cómo los SAD y la Soft Computing son técnicas con características apropiadas para ser usadas en la resolución de problemas de Bioinformática como es el caso del principal problema abordado en esta tesis: la comparación de estructuras de proteínas. En el capítulo 2 se mostraron ya varios métodos preexistentes para comparación de estructuras de proteínas, incluyendo dos algoritmos exactos para la resolución del modelo Max-CMO. No obstante, los algoritmos exactos no garantizan la finalización de su ejecución en un tiempo razonable para cualquier instancia y no pueden aplicarse, en general, para instancias constituidas por proteínas de gran tamaño. Además, los restantes algoritmos existentes de comparación de estructuras de proteínas tienen tanto sus puntos fuertes como sus puntos débiles, y su rendimiento puede variar considerablemente para cada conjunto de instancias distinto. Teniendo todo esto en consideración, la tesis se marcó como objetivo crear un nuevo algoritmo heurístico que fuera veloz y capaz de obtener buenos resultados en comparación con los restantes métodos existentes, para que pueda llegar a convertirse en una herramienta útil en la investigación sobre las proteínas.

Por tanto, en este capítulo se presenta la primera herramienta basada en

Soft Computing que se propone en esta tesis, enfocada a la resolución del modelo Max-CMO de comparación de estructuras de proteínas: un algoritmo de Búsqueda por Entornos Variables Multiarranque (MSVNS, por su nombre en inglés: MultiStart Variable Neighborhood Search). Tras la presentación del algoritmo se realizan una serie de experimentos que se orientan a probar que MSVNS puede obtener buenos resultados en términos de optimización y en términos biológicos, mediante la comprobación de que MSVNS es capaz de producir agrupamientos y ordenaciones (rankings) de similaridad apropiados sobre varios conjuntos de proteínas. La elección de la búsqueda por entornos variables (VNS) como base para el desarrollo de MSVNS se debe al buen comportamiento conocido de VNS en muchos otros problemas sobre grafos [15]. Este buen comportamiento conocido hacía prever que existían grandes posibilidades de obtener resultados valiosos con este tipo de métodos en el problema Max-CMO, dado que también es un problema definido sobre grafos. Posteriormente, veremos que estas expectativas se han visto cumplidas satisfactoriamente.

Por otro lado, hay que resaltar que el SAD SiGMAProt, que se desarrolló tomando SiGMA como base (capítulo 3), ha sido utilizado en la realización de todos los experimentos computacionales de este capítulo, de manera que todos los nuevos resultados presentados provienen de resultados obtenidos gracias al uso de SiGMAProt para el manejo y ejecución de los distintos algoritmos.

4.1. Algoritmo MSVNS de Búsqueda por Entornos Variable Multiarranque

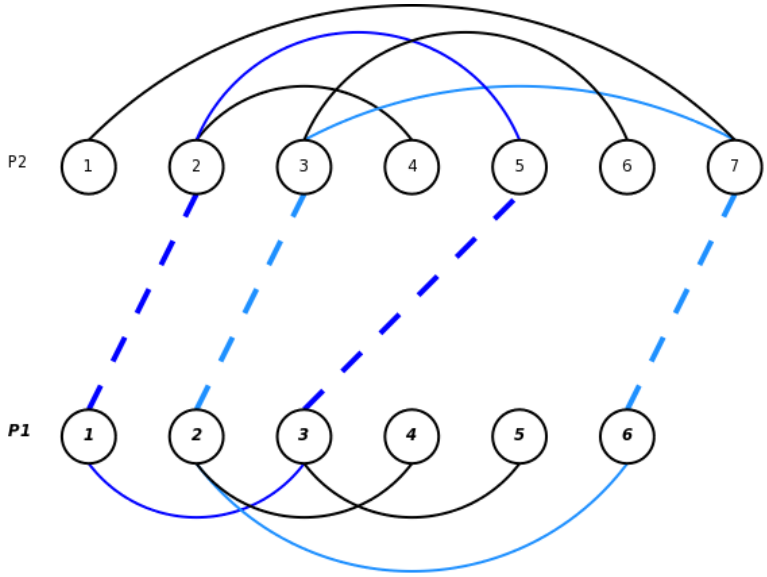
Esta sección presenta el nuevo algoritmo de Búsqueda por Entornos Variables Multiarranque (MSVNS) que se ha desarrollado para resolver el problema de máxima superposición de mapas de contacto (Max-CMO, que fue descrito en la sección 2.2.8), cuyo objetivo es la identificación de buenas soluciones mediante la utilización de diferentes estrategias para la agregación, desplazamiento y eliminación de pares dentro de las soluciones. La decisión de escoger VNS como la metaheurística de partida para resolver el modelo Max-CMO viene de la naturaleza de problema sobre grafos que tiene dicho modelo, y del hecho de que VNS ya ha sido aplicado anteriormente con éxito en la resolución de problemas sobre grafos. Entre los problemas resueltos de

forma exitosa por VNS se encuentran los problemas asociados a las redes bayesianas, el problema del descubrimiento de la teoría de grafos, los problemas de rutas como el viajante de comercio (TSP), el problema del árbol generardor mínimo, el problema del clique máximo y muchos otros.

Pasando a la descripción del MSVNS, en primer lugar, y para mayor claridad de la exposición, diremos que una solución para el problema Max-CMO (o lo que es lo mismo, un alineamiento) será representada como un vector del tamaño de la proteína más pequeña. Cada posición de este vector se corresponderá con un residuo de dicha proteína más pequeña y contendrá el residuo de la otra proteína con el que está emparejado, o un 0 en caso de que ese residuo en concreto no tenga asignada ninguna pareja. La figura 4.1 muestra un ejemplo de representación para una solución concreta. En lo que sigue numeraremos a la proteína más pequeña, y su mapa de contacto asociado, como 1, y a la proteína más grande, y su correspondiente mapa de contacto, como 2.

El algoritmo MSVNS usa, por tanto, la representación de soluciones anterior y sigue un esquema similar a un algoritmo VNS estándar típico, con la excepción de algunas modificaciones adicionales que pueden verse en el algoritmo 8. Un algoritmo VNS básico sigue normalmente el esquema que se muestra en las líneas 1-19 (excluyendo las líneas 2 y 18). El algoritmo MSVNS difiere del VNS básico en dos puntos: la incorporación de un bucle multiarranque adicional (líneas 2 y 20) para realizar múltiples reinicios (restarts), y el uso de una estrategia de simplificación (función *simplificar* en la línea 18).

La utilidad de la función *simplificar* es evitar la saturación de las soluciones (la imposibilidad de añadir más *pares* factibles a las mismas), dando así mayores oportunidades a las distintas estructuras de entornos para explorar satisfactoria y exitosamente el espacio de soluciones. Esta función se basa en la eliminación de *pares* inútiles, que no participan en ningún ciclo de longitud 4, y que, por tanto, no contribuyen al valor objetivo de la solución. Por ejemplo, en la figura 4.2, los *pares* $2 \leftrightarrow 1$ y $6 \leftrightarrow 7$ pertenecen a un ciclo, mientras que el *par* $3 \leftrightarrow 6$ no pertenece a ningún ciclo y será eliminado por la función *simplificar*, colocando un 0 en la posición 3 del correspondiente vector para la solución.



(a) Ejemplo de solución

2	3	5	0	0	7
1	2	3	4	5	6

(b) Vector para la solución de ejemplo

Figura 4.1: Vector para almacenar una solución de ejemplo.

Algoritmo 8 Algoritmo MSVNS.

procedure $MSVNS()$

1: **Seleccionar estructuras de entorno** $N_k, k = 1, ..., k_{max}$
2: **for** ($start = 0$; $start <= numStarts$; $start$++) **do**
3: $s \leftarrow$ **GenerarSoluciónInicial()**
4: **while no se cumplan las condiciones de parada do**
5: $k \leftarrow 1$
6: **while** $k < k_{max}$ **do**
7: $s' \leftarrow$ **ElegirAleatoriamente**($N_k(s)$) // **Agitación**
8: $s'' \leftarrow$ **BúsquedaLocal**(s')
9:
10: **if** $f(s'') < f(s)$ **then**
11: // **Movimiento**
12: $s \leftarrow s''$
13: $k \leftarrow 1$
14: **else**
15: $k \leftarrow k + 1$
16: **end if**
17: **end while**
18: **simplificar**(x);
19: **end while**
20: **end for**

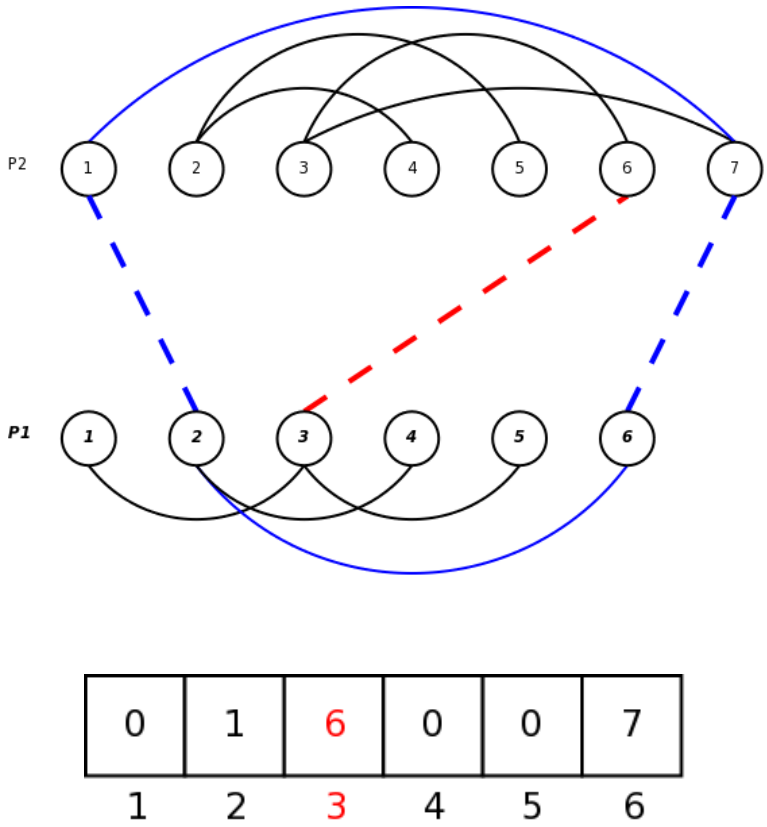


Figura 4.2: Ejemplo de la función *simplificar* - El *par* $3 \leftrightarrow 6$, formado por $6 \in P_1$ y $3 \in P_2$, será eliminado.

Pero, incluso con la presencia de la función *simplificar* y la reconstrucción parcial de soluciones que permite la estructura de entornos *neighborhoodAdd* (explicada posteriormente), es bastante frecuente que el algoritmo quede atrapado en una pequeña región del espacio de soluciones. Esta región recibirá una buena intensificación de la búsqueda, pero la diversificación de la misma puede llegar a ser muy baja, y en ello reside la motivación para la incorporación del bucle multiarranque. Por tanto, el propósito de los múltiples reinicios es evitar el problema de que la búsqueda se concentre en una región limitada del espacio de soluciones, ya que el uso de múltiples puntos de arranque ayudará a que la diversificación durante la búsqueda se vea considerablemente incrementada, explorando de forma más amplia el espacio completo de soluciones.

Se usan dos estructuras de entorno diferentes en la fase de "Agitación" del algoritmo MSVNS: una estructura de entorno que mueve aleatoriamente un *par* dentro de la solución (*neighborhoodMove*); y una estructura de entorno que añade un *par* aleatorio al alineamiento actual (*neighborhoodAdd*).

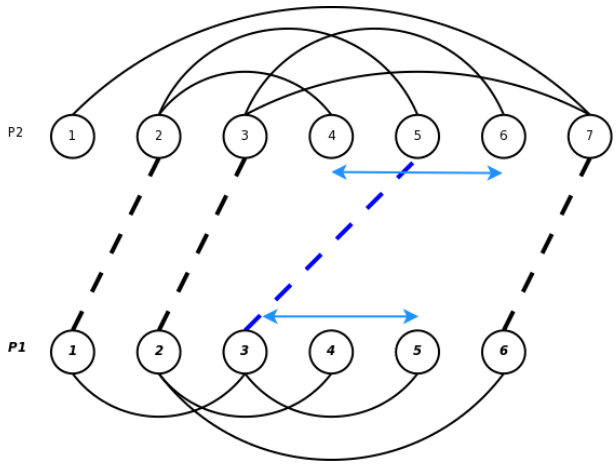
La estructura *neighborhoodMove* mueve un *par* de la siguiente manera:

1. Escoge aleatoriamente un *par* $pairCM1 \leftrightarrow pairCM2$, donde *pairCM1* es el residuo del mapa de contacto 1 que entra en el par, y *pairCM2* el correspondiente residuo en el mapa de contacto 2.

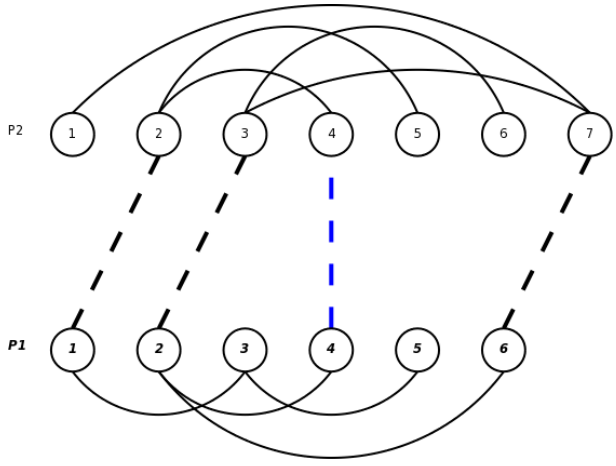
2. A continuación, se encuentran los residuos emparejados más cercanos que hay a la izquierda (*pairCM1Left*) y a la derecha (*pairCM1Right*) de *pairCM1*, así como los residuos en el mapa de contacto 2 a los que están emparejados (*pairCM2Left* y *pairCM2Right* respectivamente). Es decir, se encuentran los *pares* $pairCM1Left \leftrightarrow pairCM2Left$ y $pairCM1Right \leftrightarrow pairCM2Right$. Las componentes de estos pares marcan los intervalos en los que *pairCM1* y *pairCM2* pueden moverse sin que la solución deje de ser factible.

3. Una vez determinados los intervalos anteriores, el *par* $pairCM1 \leftrightarrow pairCM2$ inicial se reemplaza por un nuevo *par* factible $pairCM1' \leftrightarrow pairCM2'$ donde $pairCM1' \in [pairCM1Left + 1, pairCM1Right - 1]$ y $pairCM2' \in [pairCM2Left + 1, pairCM2Right - 1]$.

La aplicación de la estructura de entorno *neighborhoodMove* garantiza el mantenimiento de la factibilidad de la solución. Se muestra un ejemplo de



(a)



(b)

Figura 4.3: Ejemplo de la estructura de entorno *neighborhoodMove*. Se elige un *par* a mover y se identifican sus intervalos de movimiento factible (a). A continuación, se selecciona un *par* factible de la región delimitada por los intervalos anteriores y se sustituye con él el *par* original.

la misma en la figura 4.3, donde el *par* $3 \leftrightarrow 5$ puede moverse para pasar a ser un *par* constituido por cualquier residuo del 3 al 5 del primer mapa de contacto y cualquier residuo entre el 4 y el 6 del segundo mapa de contacto. Finalmente, se escoge el *par* $4 \leftrightarrow 4$ para sustituir al *par* original, el cual desaparece.

La estructura de entorno *neighborhoodAdd* añade un par aleatorio a la solución mediante el siguiente procedimiento:

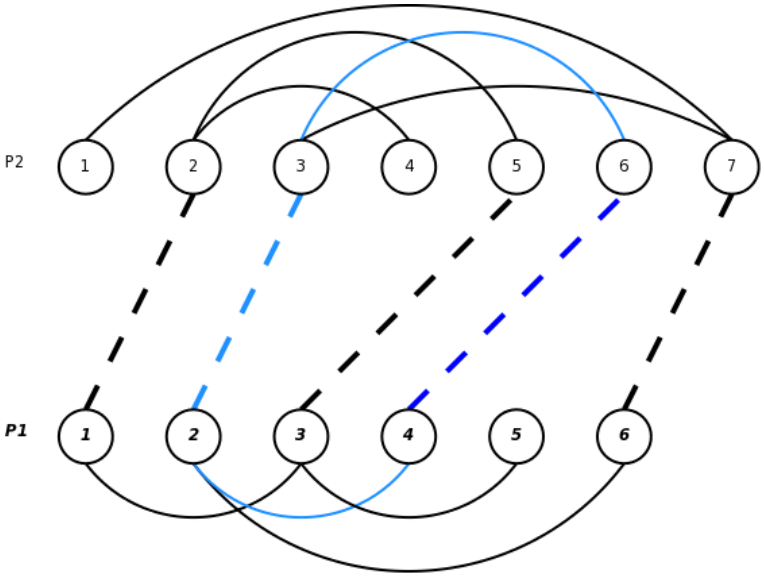
1. Se escoge un residuo aleatorio no emparejado (*pairCM1*) perteneciente al mapa de contacto 1.

2. Se encuentran *pairCM2Left* y *pairCM2Right* de la misma manera en que se hacía para *neighborhoodMove*.

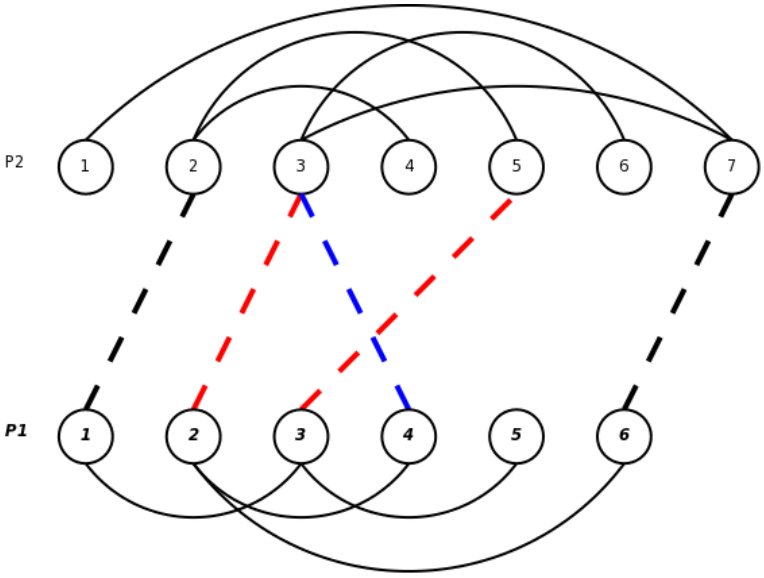
3. En lugar de simplemente emparejar *pairCM1* con un residuo del mapa de contacto 2 entre *pairCM2Left* y *pairCM2Right*, el rango de posibles parejas se expande en función del tamaño de una ventana (*window*). El nuevo *par* tendrá la forma *pairCM1* $\leftrightarrow$ *pairCM2'* donde *pairCM2'* $\in [pairCM2Left - window\ /\ 2, pairCM2Right + window/2]$. Los tamaños de ventana pueden ser especificados tanto de forma fija como mediante un porcentaje del tamaño del mapa de contacto más grande (por ejemplo, una ventana de un 10 % al comparar dos mapas de contacto de tamaños respectivos 80 y 100, se correspondería con una ventana de tamaño 10, es decir, un 10 % de 100, que es el mayor de los tamaños).

4. Dado que el *par* añadido puede producir potencialmente una solución no factible, todos los *pares* preexistentes que entran en conflicto con el nuevo *par* son eliminados. De esta forma, esta estructura de entorno añade un *par* y puede también llegar a eliminar porciones de la solución, incrementando las oportunidades de reconstruir el trozo eliminado de una forma más beneficiosa. Por tanto, cuanto mayor sea el valor del parámetro *window*, más posibilidades habrá de que se limpien trozos de la solución, abriéndose hueco para la creación de nuevos *pares* alternativos.

La figura 4.4 muestra un ejemplo de aplicación de la estructura de entorno *neighborhoodAdd* donde el residuo escogido para emparejar es el cuarto



(a)



(b)

Figura 4.4: Ejemplo de la estructura de entorno *neighborhoodAdd*. Se elige el residuo $4 \in P_1$ para ser emparejado. El nuevo *par* creado puede ser factible, como se muestra en (a), o no factible (b). En este segundo caso, la factibilidad se restaura eliminando los *pares* $(2 \in P_1, 3 \in P_2)$ y $(3 \in P_1, 5 \in P_2)$.

del mapa de contacto 1. Las restricciones de factibilidad sólo permiten su emparejamiento con el residuo número 6 del mapa de contacto 2, dando como resultado el alineamiento que se muestra en (a). Este *par* $(4 \leftrightarrow 6)$ incrementa además en 1 el valor objetivo, al formarse un nuevo ciclo con los *pares* $2 \leftrightarrow 3$ y $4 \leftrightarrow 6$. El efecto de introducir el concepto de ventana puede verse en (b), donde se permite añadir el *par* $4 \leftrightarrow 3$. Al añadir este *par*, la factibilidad debe ser restaurada, y ello se logra mediante la eliminación de los *pares* $2 \leftrightarrow 3$ y $3 \leftrightarrow 5$.

Las dos estructuras de entorno anteriores (*neighborhoodMove* y *neighborhoodAdd*) se usan para definir las estructuras de entorno del algoritmo MSVNS. Durante esta tesis, las configuraciones concretas utilizadas han sido:

- k = 1: *neighborhoodMove*.
- k = 2: *neighborhoodAdd* con un tamaño de ventana (*window*) pequeño.
- k = 3: *neighborhoodAdd* con un tamaño de ventana (*window*) intermedio.
- k = 4: *neighborhoodAdd* con un tamaño de ventana (*window*) grande.

Como se puede ver, y con objeto de conservar las propiedades e ideas de un algoritmo VNS básico, las estructuras de entorno basadas en *neighborhoodAdd* utilizan siempre valores de ventana que crecen según aumenta el valor de k. Siguiendo esta línea, los experimentos iniciales de la tesis utilizaron configuraciones de tamaños de ventana que eran independientes de las proteínas que se comparaban. Posteriormente, se realizó la modificación que permite definir los tamaños de ventana como un porcentaje del tamaño de la proteína más grande. Este cambio mejora la calidad media de las soluciones obtenidas, aunque pueden darse resultados peores en algunos pares concretos de proteínas o para algunos conjuntos de datos específicos. La sección 4.2 muestra experimentos que usan las dos aproximaciones para los tamaños de ventana, además de probarse diferentes configuraciones de valores para el caso de ventanas definidas como porcentajes. No obstante, todas las configuraciones estudiadas del MSVNS siguen la misma secuencia en cuanto a los tipos de estructura de entorno se refiere. Así, la búsqueda comienza siempre con la estructura de tipo *neighborhoodMove*. Luego, cuando MSVNS no puede mejorar el valor objetivo (valor de superposición), se incrementa el

valor de k y la búsqueda continúa con el entorno de tipo *neighborhoodAdd* y el tamaño de ventana más pequeño. Al producirse una nueva imposibilidad de mejora, la búsqueda se repite usando un nuevo entorno de tipo *neighborhoodAdd*, pero con el tamaño de ventana intermedio. Finalmente, si, estando con el tamaño de ventana intermedio, se produce un nuevo estancamiento, la búsqueda continuará con el mayor de los 3 tamaños de ventana utilizados. Si esta última estructura de entorno no puede mejorar la solución, MSVNS continúa mediante la realización de un nuevo reinicio hasta que se alcance el número total de reinicios solicitados.

La fase de búsqueda local en el algoritmo MSVNS utiliza una estructura de entorno diferente, que no se corresponde con ninguna de las anteriores. Concretamente, el proceso consiste en la realización de un bucle que recorre desde el primer residuo del mapa de contacto 1 hasta el último, y trata de emparejar cada uno de estos residuos con cada uno de los residuos factibles del mapa de contacto 2 (eliminando su emparejamiento previo si lo hubiera). Si alguna de estas tentativas de emparejamiento mejora la solución actual, se realiza el empararejamiento (primera mejora) y se vuelve a repetir el proceso con la nueva solución hasta que no es posible realizar ninguna nueva mejora.

Finalmente, el criterio de parada que se utiliza para cada reinicio del algoritmo MSVNS es que se alcancen 100 iteraciones o que hayan transcurrido 20 iteraciones sin realizar ninguna mejora (lo que ocurra antes).

Una versión inicial de este algoritmo MSVNS fue presentada en el 18th MECVNS [52] y una versión actual (con muchos de los experimentos que se incluyen más adelante en este capítulo) ha sido enviada a la revista BMC Bioinformatics [53].

4.2. Experimentos

La validación del método MSVNS propuesto se hace mediante la realización de dos tipos distintos de experimentos computacionales. El primer conjunto de experimentos evalúa MSVNS en términos de optimización. En primer lugar, una versión inicial de MSVNS se compara contra otros algoritmos heurísticos de comparación de estructuras de proteínas. Concretamente, estos algoritmos son FANS [112], LGA [146] y SGMA [76]. Los resultados de LGA y SGMA están tomados de [76] mientras que los resultados de FANS y MSVNS han sido generados mediante la realización de las respectivas com-

paraciones sobre los mismos pares de proteínas. A continuación, para una mejor evaluación de la calidad de los resultados de MSVNS en términos de optimización, se realiza un nuevo experimento que compara MSVNS con los valores óptimos de Max-CMO obtenidos por el algoritmo exacto de Xie y Sahinidis [140, 141].

El segundo conjunto de experimentos se orienta a verificar las capacidades de MSVNS en términos biológicos. En primer lugar, se comprueba cómo dos versiones de MSVNS, con diferente calidad en términos de optimización, pueden producir valores de similaridad suficientemente buenos para recuperar la agrupación en familias de SCOP de las proteínas del conjunto de datos de Skolnick obtenido de Xie y Sahinidis [140,141]. Esto se hace simplemente aplicando algoritmos estándar de agrupamiento (clustering) a las similaridades obtenidas. Finalmente, los últimos experimentos se encargan de verificar que MSVNS es capaz de producir un buen ranking (ordenación) de similaridad, evaluando su rendimiento como clasificador en los niveles de arquitectura en CATH y plegamiento en SCOP, comparando dicho rendimiento, para varios conjuntos de datos, con el obtenido por DaliLite [90] y MatAlign [5].

Las próximas secciones se dedican a explicar, en primer lugar, el análisis de Características Operativas del Receptor (ROC), que se usará para evaluar el rendimiento como clasificador de los distintos métodos en varios de los experimentos. Tras ello, se entra directamente a realizar una descripción completa de los dos conjuntos de experimentos que acaban de ser presentados.

4.2.1. Análisis de Características Operativas del Receptor (ROC)

Un modelo de clasificación (un clasificador) es una correspondencia de las instancias de un problema con un conjunto de clases/grupos. El resultado de un clasificador puede ser tanto una etiqueta de clase como un valor numérico de tipo real (como ocurrirá con los métodos de comparación de estructuras de proteínas), estableciéndose la frontera entre clases mediante la elección de un valor de umbral. Un ejemplo sencillo de este segundo caso podría ser la elección de un umbral sobre los valores de presión sanguínea para distinguir entre personas con hipertensión y personas sanas.

Cuando se trabaja con un problema de predicción de dos clases (clasifi-

		valor real		
		p	n	total
valor predicho	p'	Verdadero Positivo	Falso Positivo	P'
	n'	Falso Negativo	Verdadero Negativo	N'
	total	P	N	

Tabla 4.1: Matriz de confusión (tabla de contingencia) para los cuatros posibles resultados de un clasificador binario.

cación binaria), el resultado puede etiquetarse como una clase positiva (p) o una clase negativa (n), dando lugar a cuatro resultados posibles en el uso de un clasificador binario. Si el resultado de una predicción es p y el valor real para una instancia es también p se tiene un *verdadero positivo (true positive o TP)*; mientras que, si el valor real era n, lo que se tiene es un *falso positivo (false positive o FP)*. Del mismo modo, un *verdadero negativo (true negative)* ocurre cuando tanto la predicción como el valor real de clase son n y un *falso negativo (false negative)* es la combinación de una predicción de n con un valor real de p. La tabla 4.1 contiene una matriz de confusión que facilita la comprensión de qué es un *verdadero/falso positivo/negativo*, mostrando las combinaciones de valores reales y predichos que se asocian a cada uno de estas cuatro consideraciones.

Una Característica Operativa del Receptor (Receiver Operating Characteristic), o, simplemente, una curva ROC, es una representación gráfica de la sensibilidad (sensitivity) frente a (1− especificidad(specifity)) para un clasificador binario a medida que varía su umbral de discriminación. La curva ROC se puede representar de forma equivalente dibujando la fracción de verdaderos positivos (TPR = True Positive Rate) contra la fracción de falsos positivos (FPR = False Positive Rate) [94].

La forma que toma la curva ROC es una buena indicación visual del desempeño de un clasificador. En la figura 4.5 se puede ver un detalle del espacio de una curva ROC, cumpliéndose las siguientes propiedades:

- Si la curva ROC de un clasificador coincidiera con la línea roja segmen-

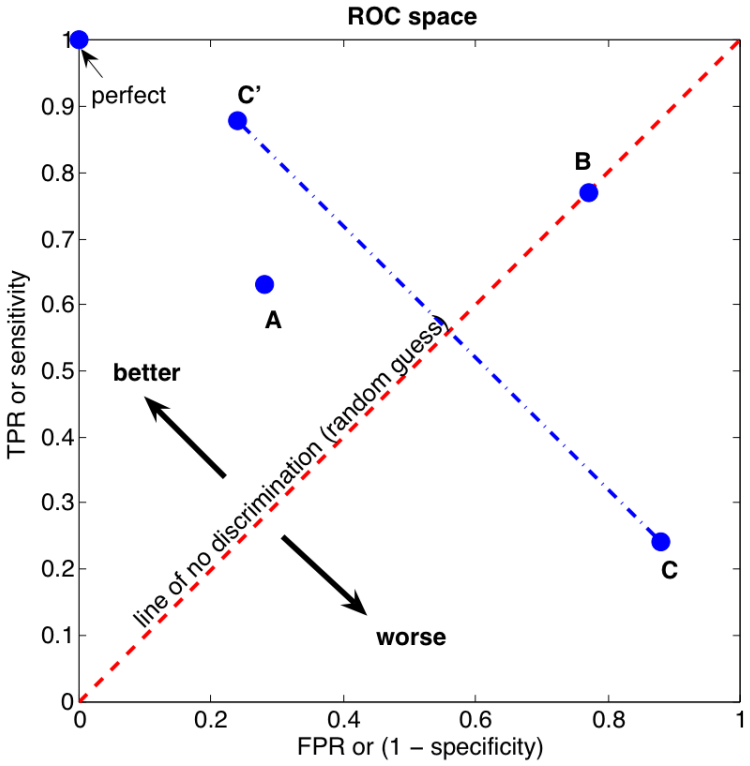


Figura 4.5: Detalle del espacio de una curva ROC.

tada (line of no discrimination) estaríamos ante un clasificador que no es capaz de discriminar las clases en absoluto, siendo equivalente a una elección de clase al azar como la realizada mediante el lanzamiento de monedas. Este es el caso del punto B en la figura, que representa un clasificador que, para una especificidad dada, es incapaz de distinguir entre las clases.

- A medida que la curva se alza por encima de la diagonal se tiene que el clasificador es mejor, tratándose de un clasificador perfecto cuando la sensibilidad toma el valor de 1 para cualquier valor de FPR o (1 - especificidad). Por ejemplo, un clasificador que conduzca al punto A tendrá unas mejores prestaciones que un clasificador que conduzca a C.

- Análogamente, el resultado de un clasificador será peor que aleatorio cuando la curva vaya por debajo de la diagonal. No obstante, cuando la curva ROC está situada bajo la diagonal, puede conseguirse un buen clasificador sin más que invertir el resultado de todas las predicciones. Este proceso convertiría puntos como el C en puntos como C', que es incluso mejor que A, pero sólo podría realizarse esta transformación si se conocieran de antemano los valores reales, así que normalmente se usarán los valores no invertidos del clasificador (como C) conduciendo ello a un clasificador de prestaciones muy pobres.

Si se genera una gráfica con dos (o más) curvas ROC para diferentes clasificadores, es posible que una de las curvas obtenga valores mayores de sensibilidad para todos los valores de especificidad. Cuando esto ocurre se puede decir claramente que el clasificador de la curva superior es mejor. Pero también puede suceder que las dos curvas se crucen en varios puntos, con trozos de cada curva alcanzando mayores valores de sensibilidad que la otra. En este caso, es mucho más difícil ver qué clasificador se comporta mejor, y ésta es la razón de que, para propósitos comparativos, se calcule un valor de resumen más sencillo: el Área Bajo la Curva (Area Under Curve o AUC). El valor de AUC es simplemente el área bajo la curva ROC y, al ser un valor numérico, permite ordenar de manera sencilla los clasificadores desde los de mayores prestaciones hasta los de menores [125].

En el problema de comparación de estructuras de proteínas, se puede usar el análisis ROC tras unos pasos muy simples. En primer lugar, se es-

cogerá como valor del clasificador el valor de similaridad que proporciona cada método o algoritmo de comparación (o una versión normalizada de dichos valores de similaridad). A continuación, puesto que en los conjunto de datos suele haber proteínas correspondientes a más de únicamente uno o dos tipos (más de dos tipos de proteínas con respecto a algún nivel de similaridad estructural), se necesita construir una clase binaria adicional. Esta nueva clase indicará si las dos proteínas que conforman cada instancia tienen valores iguales o distintos para el nivel de similaridad estructural que se quiere clasificar (clase, familia, superfamilia, etc.). De esta forma, se puede realizar ya un análisis ROC que compare el desempeño de los clasificadores (algoritmos de comparación de proteínas) en función de su capacidad de obtener resultados mayores para proteínas del mismo tipo que para proteínas de tipos distintos. En general, y por motivos de simplificación, se utilizarán siempre los valores de AUC para ordenar los algoritmos de mejores a peores prestaciones de clasificación.

4.2.2. ¿Es beneficioso el uso de MSVNS desde el punto de vista de optimización?

Esta subsección contiene el conjunto de experimentos destinados a evaluar MSVNS en términos de los valores de optimización que obtiene. El primer experimento realizado compara una versión inicial de MSVNS con otros algoritmos heurísticos de comparación de estructuras de proteínas. A continuación, en el segundo experimento, se compara MSVNS con el algoritmo exacto de Xie y Sahinidis [140, 141].

Experimento 1: Comparación con otras heurísticas en términos de optimización

El primer experimento que se realizó con el algoritmo MSVNS fue la comparación con otras técnicas heurísticas disponibles en la literatura, que, más concretamente, fueron:

- **LGA** [146]: LGA es un método diseñado para realizar la comparación de estructuras de proteínas o fragmentos de las mismas en modos tanto dependientes como independientes de la secuencia. El programa genera muchas superposiciones locales diferentes con vistas a detectar regiones similares entre las proteínas. La función de puntuación de

LGA tiene dos componentes, LCS (segmentos continuos más largos) y GDT (test global de distancia), formulados para la detección de regiones de similaridad local y global entre las proteínas. Al comparar dos estructuras, el procedimiento LCS es capaz de localizar y superponer los segmentos más largos de residuos que se mantienen por debajo de un umbral de RMSD. El procedimiento GDT está diseñado para complementar las evaluaciones hechas por LCS buscando el conjunto más largo (no necesariamente continuo) de residuos "equivalentes" que no se separan en más de una distancia límite especificada.

LGA está disponible en la red en http://predictioncenter.org/local/lga/lga.html.

De todos los algoritmos evaluados, LGA es el único que no resuelve el modelo Max-CMO, pero sí que produce un alineamiento de los residuos de ambas proteínas. Este alineamiento puede transformarse fácilmente en un valor de superposición (en términos de Max-CMO) mediante el cálculo de la superposición asociada a dicho alineamiento en los mapas de contacto de las proteínas comparadas.

- **FANS** [112]: FANS es una metaheurística de búsqueda por entornos adaptativa y difusa (Fuzzy Adaptive Neighborhood Search). Se trata de un algoritmo de búsqueda por entornos en el que la estructura de entorno usada para recorrer el espacio de soluciones (el espacio de todas las posibles superposiciones o alineamientos) es un conjunto difuso que se ajusta de forma sistemática y dinámica durante el transcurso de la búsqueda. Al contrario que un VNS básico, FANS proporciona un segundo método para escapar de los óptimos locales y continuar la búsqueda en otras regiones prometedoras del espacio de búsqueda. Para lograrlo, FANS usa una función objetivo de tipo difuso, que define cuáles entre todas las soluciones vecinas se consideran "aceptables" para continuar en ellas la exploración. La versión actual de FANS explora las soluciones vecinas de manera secuencial y continúa la búsqueda por la primera solución aceptable que se encuentre entre ellas. La implementación utilizada en los experimentos está disponible en: http://decsai.ugr.es/~dpelta/GMAXFCMO/index.html. El algoritmo tiene un parámetro de control *lambda* que en los experimentos realizados ha recibido el valor de 1. El efecto de este ajuste es evi-

tar que FANS acepte soluciones peores para desplazarse durante la búsqueda, lo que se observó que conducía a mejores resultados en el problema de comparación.

- **SGMA** [76]: SGMA es un algoritmo memético autogenerado (Self-Generating Memetic Algorithm) que es capaz de hacer evolucionar de forma concurrente tanto las soluciones (alineamientos de proteínas) como el operador de movimiento de la búsqueda local que se necesita para resolver la instancia actual. Se afirma que la generación concurrente de la estrategia de búsqueda local y las soluciones permite que el algoritmo memético obtenga mejores resultados que los obtenidos por un algoritmo genético o por un algoritmo memético cuyas búsquedas locales hayan sido diseñadas por una persona.

Aunque estas técnicas pueden no representar las "mejores" estrategias de resolución del problema, sí que nos sirven para obtener una buena indicación de lo idoneidad de nuestro método.

Dado que este experimento se realizó en las fases iniciales de desarrollo de MSVNS, solamente se evaluó una versión sencilla llamada **MS10VNS**. **MS10VNS** corresponde al algoritmo MSVNS haciendo uso de 10 reinicios, 1 estructura de entorno de tipo *neighborhoodMove* y 3 estructuras de entorno de tipo *neighborhoodAdd* con ventanas de tamaño fijo 10, 20 y 30 respectivamente. Si se hubieran utilizado las versiones de MSVNS más avanzadas que se presentan en los experimentos posteriores, se podrían esperar resultados aún mejores, pero se ha decidido mantener la versión **MS10VNS** para mostrar lo competitivo que era MSVNS incluso en sus primeras versiones.

Como conjunto de datos para realizar esta comparación se utilizaron varias proteínas del PDB [14]: 1aa9, 1ct9, 1cnp, 1eca, 1gnp, 1hnf, 1jhg, 1qra, 5p21 y 6q21. A partir de los ficheros de definición de estas proteínas se generaron los correspondientes mapas de contacto usando $\Re = 6{,}5$ Angstroms de umbral (del mismo modo que en [76]), considerando solamente los átomos C_α de los residuos.

Tras ello, se crearon instancias formadas por los pares de proteínas presentes en los ejemplos usados en las tablas 5 y 6 de [76]. Para cada par se realizaron 5 ejecuciones de **MS10VNS** y **FANS**, conservándose la mejor solución encontrada. Los valores de superposición correspondientes a**SGMA** y **LGA** se tomaron directamente de [76].

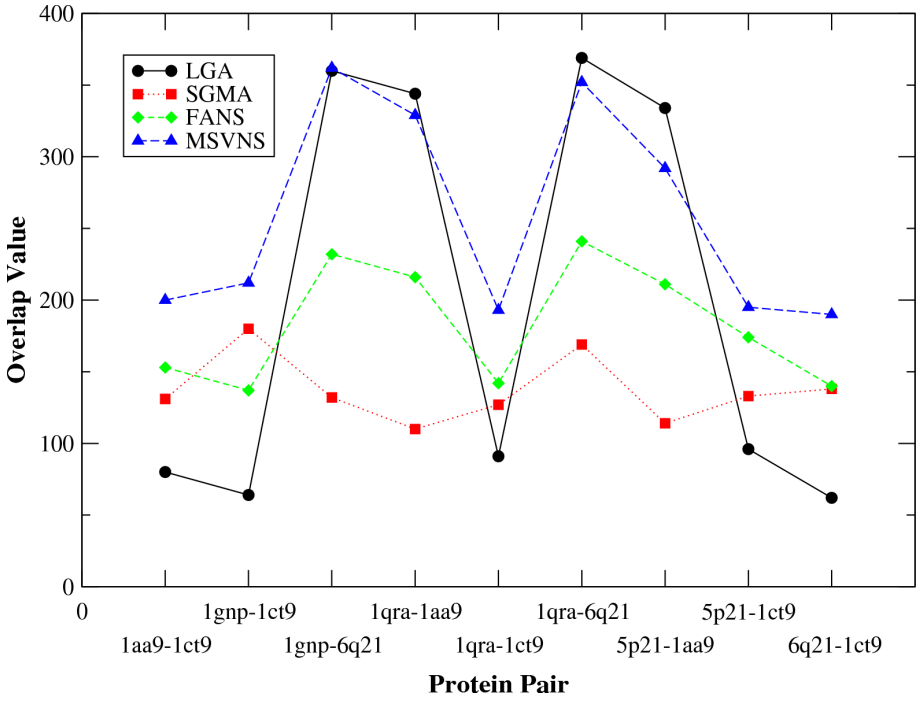


Figura 4.6: Resultados de superposición para proteínas alpha-beta. Los valores de LGA y SGMA han sido tomados de la tabla 5 de [76].

El conjunto completo de resultados se muestra como gráficas en las figuras 4.6 y 4.7. El eje X muestra los pares de proteínas comparados (indicados por su PDB code) mientras que en el eje Y se muestran los valores objetivos alcanzados. Cada serie se corresponde con los resultados de un algoritmo particular y, cuanto más altos son los valores, mejor es el alineamiento obtenido.

Las proteínas utilizadas en la figura 4.6 pertenecen a la clase "alpha-beta" y los tamaños de los mapas de contacto asociados están en torno a 170 átomos, a excepción de la proteína 1ct9 que tiene un total de 498 átomos. Como puede verse en la figura, **MS10VNS** sólo es superado por cualquiera de las otras heurísticas en 3 casos (de un total de 9), correspondientes a los pares 1qra-1aa9, 1qra-6q21 y 5p21-1aa9, donde el algoritmo **LGA** es capaz de obtener resultados mejores, aunque por muy poca diferencia. Para todos los demás pares, los mejores valores son siempre los obtenidos por

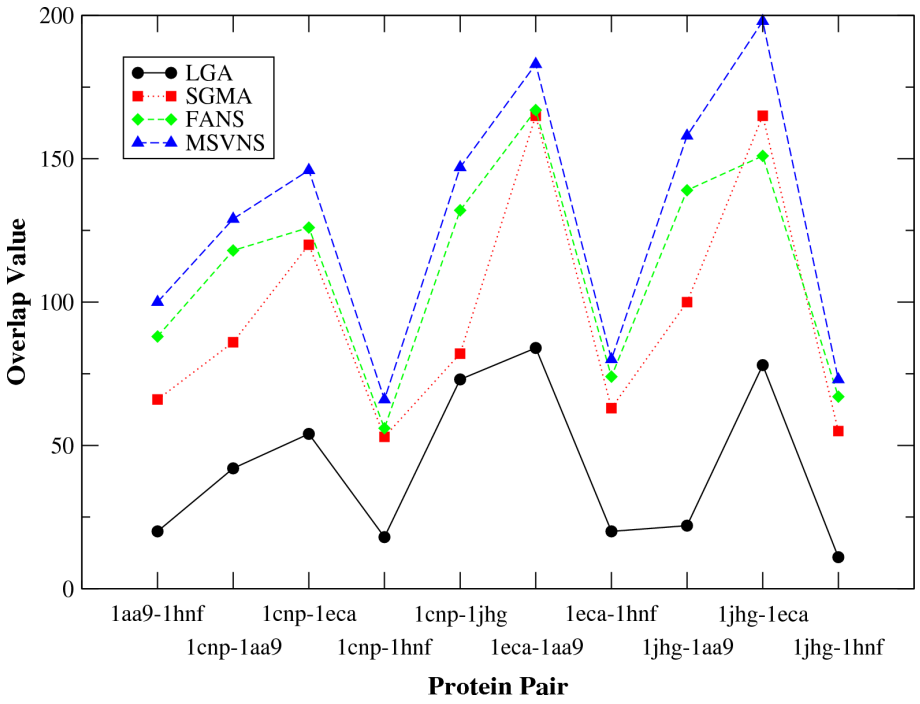


Figura 4.7: Resultados de superposición en un conjunto mixto de proteínas. Los valores de LGA y SGMA han sido tomados de la tabla 6 de [76].

			Residuos			Contactos		
Conjunto	Proteínas	Pares	Mín.	Med.	Máx.	Mín.	Med.	Máx.
Skolnick	40	161	97	158,23	255	265	470,93	815
Lancia	269	2702	44	57,07	68	33	95,91	137

Tabla 4.2: Información de los conjuntos de datos de Skolnick y Lancia.

MS10VNS. Si se miran sólo los restantes algoritmos no se puede apreciar ninguna ventaja clara de uno sobre los otros.

Las proteínas utilizadas en la figura 4.7 pertenecen a diferentes clases de proteínas y el tamaño de sus mapas de contacto asociados varía entre los 90 y los 180 átomos. Se aprecia claramente en la figura como **MS10VNS** supera a las restantes heurísticas para todos y cada uno de los pares considerados. En este conjunto de datos, el segundo mejor algoritmo es **FANS**, que también puede considerarse como un algoritmo similar a VNS, justificándose nuevamente el acierto en el uso de VNS como base de los algoritmos para la resolución del modelo Max-CMO.

Experimento 2: Comparación con los algoritmos exactos

Como punto esencial para evaluar la capacidad de MSVNS en términos de optimización, se realizó un experimento para comparar la implementación de MSVNS frente a los resultados exactos de Max-CMO de [140].

Los conjuntos de prueba utilizados en este experimento fueron dos conjuntos de datos descritos inicialmente en [78]: a) Lancia, con 269 proteínas y 2702 pares resueltos de forma óptima; y b) Skolnick, con 40 proteínas y 161 pares resueltos de forma óptima. La tabla 4.2 muestra información de los tamaños de las proteínas en ambos conjuntos.

Para realizar una comparación justa y reproducible, los mapas de contacto para todas las proteínas de ambos conjuntos de datos fueron tomados de los ficheros disponibles en http://eudoxus.scs.uiuc.edu/cmos/cmos.html. Estos mapas se basan en el uso de átomos C_α y tanto dichos ficheros como los valores de superposición óptimos fueron amablemente proporcionados por los autores de [140].

Los experimentos se han limitado a aquellos pares de proteínas para los que el algoritmo exacto fue capaz de alcanzar el óptimo en el plazo de 10 días [140]. Hay que resaltar que los experimentos de [140] se realizaron

en 3 ordenadores con procesadores de 3.0 Ghz y 1 Gb de RAM cada uno, mientras que nuestros experimentos usaron solamente una máquina con un procesador de 2.2 Ghz (AMD Athlon 64 3500+) y 1Gb de RAM. Xie y Sahinidis mejoraron posteriormente sus resultados de [140] en términos de los recursos computacionales requeridos [141], pero nuestros requisitos siguen siendo mucho más bajos, como veremos luego.

En este experimento se utilizaron 3 versiones de MSVNS, correspondientes a 3 configuraciones diferentes de parámetros para los tamaños de las ventanas en las estructuras de entorno de tipo *neighborhoodAdd*:

- MSVNS1: 1 estructura de entorno de tipo *neighborhoodMove* y 3 estructuras de entorno de tipo *neighborhoodAdd*, con tamaños de ventana respectivos del 5 %, 10 % y 15 %.
- MSVNS2: 1 estructura de entorno de tipo *neighborhoodMove* y 3 estructuras de entorno de tipo *neighborhoodAdd*, con tamaños de ventana respectivos del 10 %, 20 % y 30 %.
- MSVNS3: 1 estructura de entorno de tipo *neighborhoodMove* y 3 estructuras de entorno de tipo *neighborhoodAdd*, con tamaños de ventana respectivos del 10 %, 30 % y 50 %.

La configuración de estas 3 versiones de MSVNS presenta una diferencia importante con la versión MS10VNS del experimento previo: en lugar de utilizar tamaños de ventana fijos, los valores para cada instancia se calculan como un porcentaje del tamaño del mapa de contacto más grande en dicha instancia. De esta forma, los tamaños de las ventanas se ajustan mejor al tamaño real de las proteínas comparadas, pues no es práctico o aconsejable usar ventanas muy grandes para proteínas pequeñas o ventanas muy pequeñas cuando las proteínas a comparar son grandes. Los porcentajes ayudan a obtener valores apropiados de ventanas para cada instancia y, gracias a ello, los resultados finales se ven considerablemente mejorados. Además, para mejorar aún más los resultados, en lugar de hacer solamente 5 ejecuciones de 10 reinicios para cada instancia, las comparaciones realizadas en este experimento utilizan 1 ejecución de 150 reinicios (100 reinicios más). Hay que indicar que el número de reinicios es el factor más importante en lo que a repeticiones del algoritmo se refiere, y que no hay diferencias apreciables en dividir el número total de reinicios en varias ejecuciones o hacer

			Error(%)		
	Versión	N	Med.	Desv. est.	Mediana
Total	MSVNS1	2702 (100 %)	5,8765	7,12280	1,9049
	MSVNS2	2702 (100 %)	3,9959	5,60979	0,0000
	MSVNS3	2702 (100 %)	3,5671	5,21332	0,0000
Pares	MSVNS1	1259 (46,60 %)	0,0000	0,00000	0,0000
óptimos	MSVNS2	1522 (56,33 %)	0,0000	0,00000	0,0000
	MSVNS3	1577 (58,37 %)	0,0000	0,00000	0,0000
Pares	MSVNS1	1443 (53,40 %)	11,0037	6,21068	11,1111
no óptimos	MSVNS2	1180 (43,67 %)	9,1499	4,98958	9,0909
	MSVNS3	1125 (41,63 %)	8,5674	4,73640	8,3333

Tabla 4.3: Resultados sobre los 2702 pares del conjunto de datos de Lancia.

			Error(%)		
	Versión	N	Med.	Desv. est.	Mediana
Total	MSVNS1	161 (100 %)	7,3950	7,44111	5,5556
	MSVNS2	161 (100 %)	1,8235	2,50117	0,7375
	MSVNS3	161 (100 %)	1,6744	2,39488	0,4399
Pares	MSVNS1	42 (26,09 %)	0,0000	0,00000	0,0000
óptimos	MSVNS2	62 (38,51 %)	0,0000	0,00000	0,0000
	MSVNS3	68 (42,24 %)	0,0000	0,00000	0,0000
Pares	MSVNS1	119 (73,91 %)	10,005	6,98169	9,5092
no óptimos	MSVNS2	99 (61,49 %)	2,9655	2,60624	2,2124
	MSVNS3	93 (57,76 %)	2,8987	2,52732	2,3910

Tabla 4.4: Resultados sobre los 161 pares del conjunto de datos de Skolnick.

todos los reinicios en una única ejecución. La única diferencia práctica es que MSVNS devuelve una solución por cada ejecución completa, y no por cada reinicio individual, con lo que, al hacer todos los reinicios en una única ejecución, solamente se conserva la mejor solución. De todas formas, esto no es ningún problema para este experimento, dado que el objetivo consiste en obtener solamente los mejores valores de superposición para cada instancia.

Para cada ejecución se realiza una medida del $error(\,\%)$ con respecto al valor óptimo de la instancia. Los resultados se muestran en las tablas 4.3 y 4.4.

El principal aspecto a notar de ambas tablas es que, a medida que el

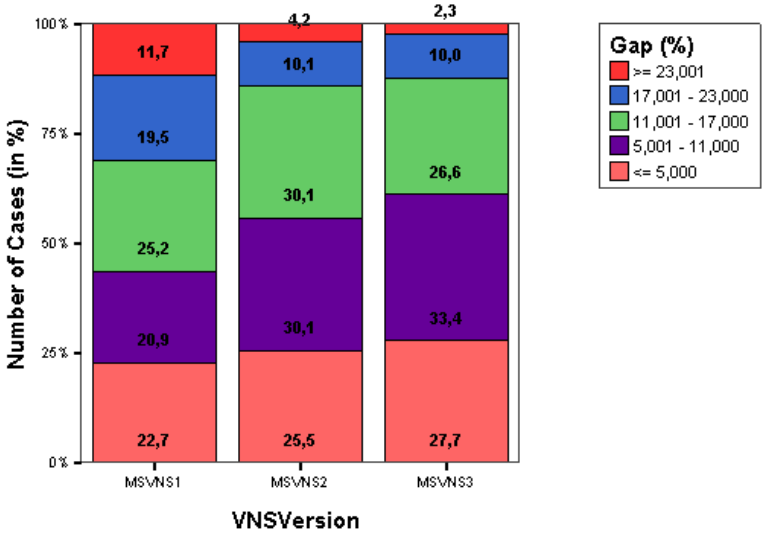


Figura 4.8: Distribución del error (gap) sobre los valores exactos (en %) para el conjunto de instancias no resueltas de forma óptima por MSVNS en el conjunto de datos de Lancia.

número de versión de MSVNS aumenta (a medida que los tamaños de ventana aumentan), el error medio decrece. La mejor alternativa es MSVNS3 con tamaños de ventana de 10 %-30 %-50 %, que conduce a un error medio por debajo del 3.6 % en los 2702 pares del conjunto de datos de Lancia, y un error medio por debajo del 1.7 % para los 161 pares de Skolnick. Puesto que los valores de mediana son mucho más bajos que los de media, las tablas también muestran el número de pares que fueron resueltos con el valor óptimo, así como los pares en los que no se alcanzó dicho óptimo. En el conjunto de datos de Lancia, casi un 60 % de los pares se resolvieron de forma óptima, mientras que, en Skolnick, el porcentaje de instancias resueltas optimamente estuvo en torno al 40 %. Nuevamente, los porcentajes de pares no resueltos optimamente disminuyen a medida que los tamaños de ventana aumentan.

Es interesante analizar también el subconjunto de los pares no resueltos de forma óptima. La figura 4.8 muestra la distribución de dichos pares en el conjunto de datos de Lancia para 5 diferentes rangos de porcentaje de error

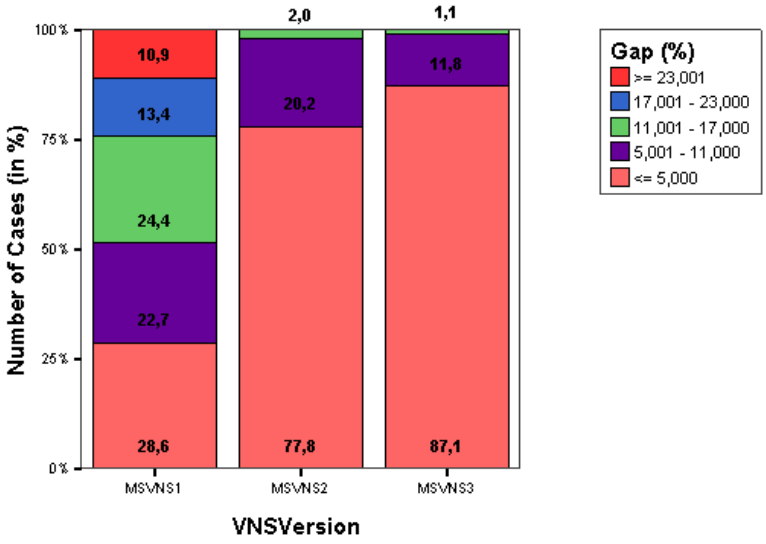


Figura 4.9: Distribución del error (gap) sobre los valores exactos (en %) para el conjunto de instancias no resueltas de forma óptima por MSVNS en el conjunto de datos de Skolnick.

	Time	
Versión	Lancia	Skolnick
MSVNS1	4 hs. 12 m.	3 hs. 11 m.
MSVNS2	4 hs. 43 m.	4 hs. 48 m.
MSVNS3	5 hs. 7 m.	5 hs. 44 m.

Tabla 4.5: Tiempos totales empleados por cada una de las versiones de MSVNS para comparar los pares de proteínas con valores exactos disponibles (2702 pares en el conjunto de datos de Lancia y 161 en el de Skolnick).

(gap) para cada una de las tres versiones de MSVNS. La figura 4.9 muestra una gráfica equivalente para el conjunto de datos de Skolnick. Ambas figuras muestran nuevamente cómo la calidad de los resultados se incrementa a medida que se pasa de la versión MSVNS1 hasta la MSVNS3 donde tanto el número de pares no resueltos de forma óptima como el error correspondiente son menores.

Algunos otros datos significativos con respecto al error en el subconjunto de pares no resueltos de forma óptima son que, en el conjunto de datos de Lancia, dicho error se encuentra por debajo del 11 % para aproximadamente la mitad o más de los pares en todas las versiones de MSVNS; y que, en el caso de MSVNS3, se tienen errores menores al 17 % para el 87 % de los pares no resueltos de forma óptima. En Skolnick, MSVNS3 es capaz de obtener para el 87 % de los pares no resueltos de forma óptima un error inferior al 5 %. Todos estos datos no hacen más que redundar en la excelente capacidad de MSVNS para acercarse a la calidad de los algoritmos exactos en una fracción del tiempo de cómputo necesario.

Con respecto a los recursos computacionales que han sido necesarios para alcanzar estos resultados, la tabla 4.5 refleja el tiempo total de cada versión de MSVNS. Es evidente que cuanto más grandes son los tamaños de ventana, mayor es el tiempo total. Esto es así porque, a medida que los tamaños crecen, el número potencial de pares alineables aumenta, conduciendo a un aumento esperado del tiempo de ejecución, aunque consideramos que el balance obtenido entre la calidad de las soluciones y el tiempo requerido es más que razonable.

En la aproximación presentada en [141], los autores mejoraron el tiempo total de [140] estableciendo límites de tiempo por instancia que variaban desde 4 segundos a 10-30 minutos o incluso más. Esta estrategia requirió un

día y medio para resolver de forma óptima 1233 pares del conjunto de datos de Lancia. Se necesitaron 3 días para alcanzar 1997 instancias resueltas de forma óptima y 9 para lograr la cifra final de 2702 instancias resueltas óptimamente, haciendo uso de un solo equipo. Si los comparamos con los valores para MSVNS de las tablas 4.3 y 4.5, la peor versión (MSVNS1) necesitó aproximadamente 4 horas para resolver las 2702 instancias, alcanzando el óptimo en 1259 de ellas. Con menos de una hora de cómputo más, MSVNS3 fue capaz de aumentar el número de óptimos hasta 1577. Por desgracia, no están disponibles los tiempos de ejecución de los algoritmos exactos para el conjunto de datos de Skolnick. Su disponibilidad permitiría evaluar el comportamiento en proteínas de mayor tamaño (ver tabla 4.2), en las que las diferencias de MSVNS con los exactos, en cuanto a tiempos, podrían ser incluso mayores, dada la naturaleza NP-completa del problema.

Hay que considerar, también, que los tiempos empleados por MSVNS pueden controlarse de una forma bastante sencilla. En primer lugar, el uso de una u otra versión de MSVNS conduce a variaciones en la relación calidad/tiempo de las soluciones, permitiendo que se escoja el mejor balance para cada situación. En segundo lugar, la velocidad de MSVNS está relacionada de manera lineal con el número de reinicios, de forma que se pueden conseguir fácilmente mejoras en el tiempo mediante el establecimiento de un menor valor para el número de reinicios, aunque los resultados también se verían degradados.

Los resultados de los dos experimentos presentados confirman que la estrategia MSVNS es útil para resolver el modelo Max-CMO, con valores cercanos al óptimo en la gran mayoría de todos los pares de proteínas evaluados.

4.2.3. ¿Puede proporcionar MSVNS un ranking de similaridad de proteínas apropiado?

Aunque el análisis desde el punto de vista de optimización, que se acaba de realizar, es importante y relevante, también es interesante evaluar la calidad de MSVNS como clasificador de estructuras de proteínas. En efecto, hasta donde alcanza nuestro conocimiento, incluso el papel de Max-CMO para realizar agrupamientos y clasificación ha sido estudiado previamente sólo en el nivel de familia en SCOP para el conjunto de datos de Skolnick. Por ello, en esta tesis vamos a realizar un análisis de la capacidad de nuestra

estrategia para proporcionar valores de superposición (normalizados) que ofrezcan un ranking (ordenación) de similaridad apropiado para realizar una clasificación en otros niveles de similaridad estructural.

Para la realización de estas tareas, es más importante obtener valores de superposición que reflejen las similaridades estructurales dentro de un conjunto de proteínas que tener los mejores valores de superposicion posibles para pares concretos de proteínas. Esto no quita que, ante la ausencia de otras diferencias, si dos algoritmos obtienen un buen ranking de similaridad, aquél que produzca los valores de superposición más altos será el algoritmo preferido. En otras palabras, se va a comprobar cómo no es realmente necesario resolver Max-CMO de forma óptima para realizar clasificación a nivel estructural, pues los valores subóptimos pueden ser suficientemente buenos.

Más aún, los valores de superposición pueden no ser adecuados por sí mismos con el propósito de clasificación porque dichos valores dependen del tamaño de las proteínas que están siendo comparadas. Esto no es un problema si las proteínas que se comparan tienen todas tamaño similar, pero podría ser muy significativo en el caso de que las proteínas en el conjunto de datos presenten diferencias de tamaño importantes. Por tanto, se va a estudiar también la aplicación de esquemas de normalización, y se mostrará cómo pueden llegar a jugar un papel crucial en la clasificación. A pesar de la ausencia de un criterio general de cómo se debe realizar la normalización, hay al menos tres alternativas basadas en las existentes en la literatura:

$$Norm1(P_i, P_j) = overlap(P_i, P_j)/min(contacts\ P_i, contacts\ P_j) \tag{4.1}$$

$$Norm2(P_i, P_j) = 2 * overlap(P_i, P_j)/(contacts\ P_i + contacts\ P_j) \tag{4.2}$$

$$Norm3(P_i, P_j) = \begin{cases} 0 & \text{si la diferencia de contactos} \\ & \text{es superior a un } 75\,\% \\ Norm1(P_i, P_j) & \text{en otro caso} \end{cases} \tag{4.3}$$

Las dos primeras alternativas fueron propuestas en [79] y [140] respectivamente. Aquí proponemos la tercera, cuyo objetivo es evitar la comparación de dos estructuras cuyos tamaños son totalmente diferentes.

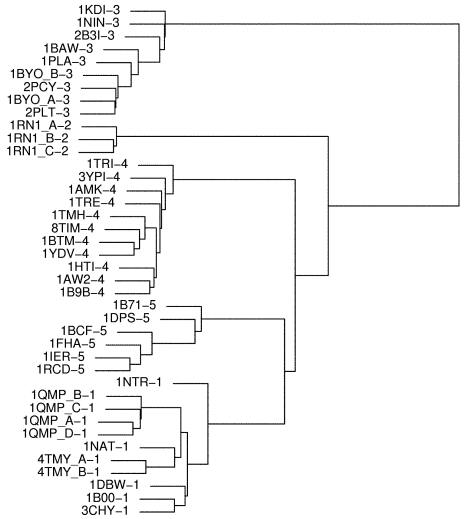
Se han realizado tres experimentos para analizar las propuestas anteriores. En el primero de ellos, se realiza una comparación de tipo todos contra todos sobre el conjunto de datos de Skolnick para comprobar la capacidad

de la aplicación de un algoritmo de agrupamiento para discriminar entre diferentes clases de proteínas a partir de los valores de similaridad obtenidos. En el segundo experimento, se evalúa el rendimiento de nuestra estrategia para detectar similaridad en el nivel de plegamiento en SCOP haciendo uso del conjunto de datos de Fischer [39]. Para este experimento se realiza asimismo una comparación con DaliLite [90]. Finalmente, se realizan una serie de consultas sobre la base de datos NH3D [131] para evaluar la capacidad de MSVNS de detectar similaridades al nivel de arquitectura en CATH. En este último experimento se realizan comparaciones tanto con DaliLite [90] como con MatAlign [5].

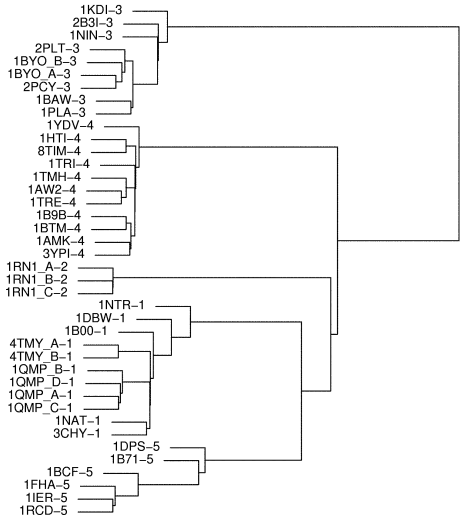
Experimento 1: Ranking de similaridad en el conjunto de datos de Skolnick

Para este experimento se usa nuevamente el conjunto de datos de Skolnick tomado de [140,141], que es una versión revisada de la versión presentada en [20], donde cada proteína se etiqueta con una clase que se corresponde con una de entre cinco familias diferentes de la base de datos SCOP [105]. El procedimiento realizado ha consistido en la comparación de todos los pares de proteínas usando tanto la mejor versión de nuestra estrategia desde el punto de vista de optimización (MSVNS3) como la peor (MSVNS1). A continuación, se ha construido una matriz de similaridad para cada versión de MSVNS usando los valores de superposición normalizados con $Norm1$, $Norm2$ y $Norm3$. Finalmente, se aplicaron sobre cada una de las matrices dos algoritmos de agrupamiento jerárquico implementados en el paquete estadístico R [116]: agrupamiento por enlace simple y agrupamiento por enlace promedio (single y average linkage clustering en su nomenclatura inglesa). El propósito de estos pasos es evaluar si la agrupación en clases original puede recuperarse a través de los valores de superposición normalizados o no.

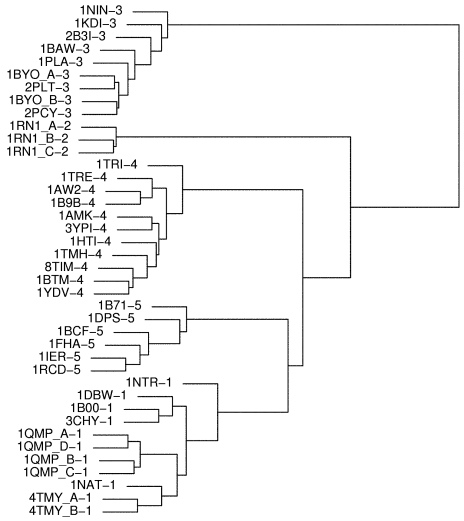
Tanto MSVNS1 como MSVNS3 son capaces de recuperar perfectamente la agrupación original independientemente del esquema de normalización y de agrupamiento que se aplique. La figura 4.10 muestra algunos ejemplos de estos resultados para el caso en que los agrupamientos por enlace simple y por enlace promedio se aplicaron sobre las matrices de similaridad normalizadas con *Norm1*, obteniéndose resultados similares con las otras



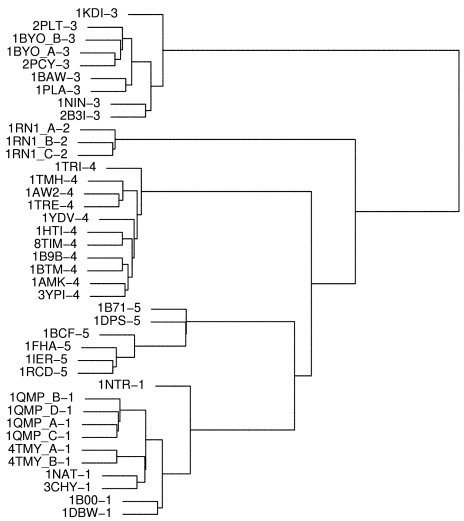
(a) MSVNS1, Enlace simple



(b) MSVNS3, Enlace simple



(c) MSVNS1, Enlace promedio



(d) MSVNS3, Enlace promedio

Figura 4.10: Agrupamiento jerárquico basado en los valores de superposición normalizados (usando $Norm1$) de todos los pares de proteínas en el conjunto de datos de Skolnick. Los dendongramas superiores (a, b) corresponden a agrupamiento por enlace simple y los inferiores (c, d) a agrupamiento por enlace promedio. Por otro lado, los dendogramas a la izquierda (a, c) corresponden a valores provenientes de MSVNS1 mientras que los dendogramas a la derecha (c, d) corresponden a resultados de MSVNS3.

			Residuos			Contactos		
Conjunto	Proteínas	Pares	Mín.	Med.	Máx.	Mín.	Med.	Máx.
Fischer	68	4624	56	211,16	581	147	636,66	1952

Tabla 4.6: Información del conjunto de datos de Fischer.

normalizaciones. El número de clase se muestra a la derecha de cada uno de los identificadores de las proteínas con objeto de facilitar la visualización.

En [141] se reconocieron exactamente los mismos grupos haciendo uso del algoritmo exacto para el Max-CMO. Por tanto, nuestro estudio ha mostrado que nuestra estrategia MSVNS puede replicar los resultados obtenidos mediante los métodos exactos haciendo uso de una estrategia más sencilla y menos recursos computacionales. De manera adicional, hemos confirmado que se puede conseguir una clasificación correcta usando valores de Max-CMO no óptimos convenientemente normalizados. Ambos resultados son resultados importantes por sí mismos.

Experimento 2: Evaluando la similaridad en el conjunto de datos de Fischer

Se muestra a continuación otro experimento realizado haciendo uso del conjunto de datos de Fischer (descrito en la tabla II de [39]), que está compuesto de 68 proteínas pertenecientes a diferentes clases y plegamientos. La tabla 4.6 proporciona información de los tamaños de las proteínas en este conjunto de datos.

Se realizó una comparación todos contra todos de las proteínas del conjunto de datos haciendo uso de MSVNS2 y DaliLite [90] con sus parámetros por defecto. En este caso, el objetivo era analizar la capacidad de los dos métodos para reconocer similaridades estructurales al nivel del plegamiento en SCOP. Se escogió la versión MSVNS2 del algoritmo MSVNS sobre las versiones MSVNS1 (más rápida) y MSVNS3 (mejores soluciones) porque consideramos que MSVNS2 proporciona un buen balance entre la calidad de las soluciones y la velocidad. Además, aunque MSVNS3 sea mejor que MSVNS2 en términos de optimización, el impacto de usar MSVNS2 frente a MSVNS3 en términos de clasificación es de esperar que sea mucho menor. La razón para ello es que, pese a que todos los valores obtenidos con MSVNS2 tenderán a ser menores que los que pueda alcanzar MSVNS3, esta

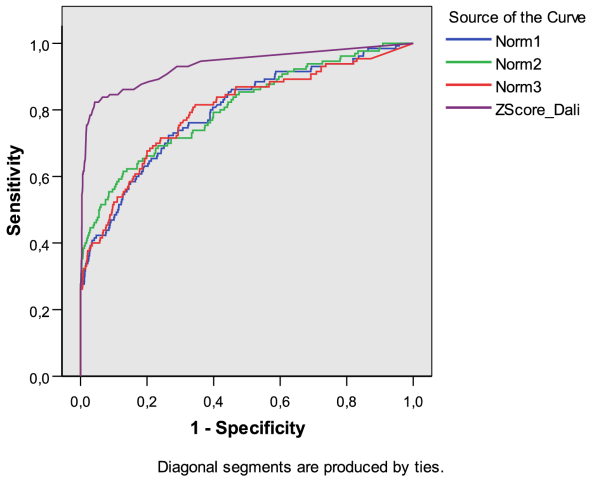


Figura 4.11: Curva ROC para cada medida sobre el conjunto de datos de Fischer en la clasificación a nivel del plegamiento en SCOP.

			Intervalo de confianza asintótico al 95 %	
Medida	Área	Error típ.	Límite inferior	Límite superior
$Norm1$	0,795	0,016	0,765	0,826
$Norm2$	0,805	0,016	0,774	0,836
$Norm3$	0,797	0,016	0,765	0,830
$DaliLiteZScore$	0,933	0,011	0,912	0,954

Tabla 4.7: Valores de AUC para cada medida sobre el conjunto de datos de Fischer para la clasificación a nivel de plegamiento en SCOP.

degradación de resultados tenderá a mantenerse de forma similar para todas las instancias. Ello implica que la relación y la ordenación (ranking) del conjunto completo de valores de superposición se conservará en la mayoría de los casos. Hemos visto también que esto es así en el experimento previo, donde tanto MSVNS1 como MSVNS3 eran capaces de producir valores de superposición que condujeron a agrupamientos perfectos. Para las comparaciones realizadas se han tomado como medidas (clasificadores) los valores de Z-Score devueltos por DaliLite y los valores resultantes de aplicar los tres esquemas de normalización ($Norm1$, $Norm2$ y $Norm3$) sobre los valores de superposición de MSVNS2.

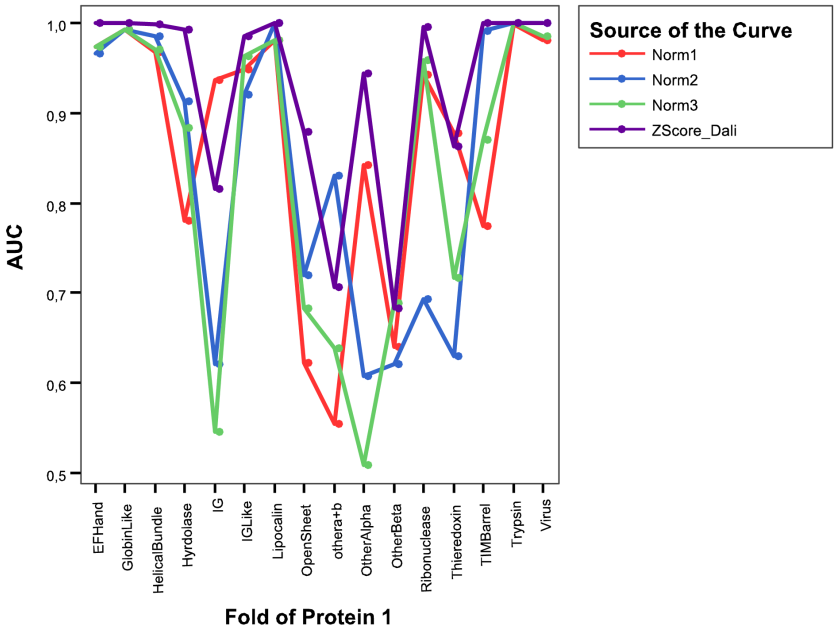


Figura 4.12: Valores de AUC para cada medida sobre el conjunto de datos de Fischer en la clasificación a nivel del plegamiento en SCOP diferenciando según los distintos plegamientos.

La figura 4.11 muestra las curvas ROC globales para la clasificación a nivel de plegamiento, mientras que la tabla 4.7 muestra los correspondientes valores de AUC calculados con ©SPSS 14.0. Se puede observar que DaliLite alcanza, con diferencia, los valores de AUC más altos.

Sin embargo, si diferenciamos los valores de AUC en función del plegamiento, tal y como se muestra en la figura 4.12, se observan dos hechos destacables que conviene resaltar. En primer lugar, se puede observar que para algunos plegamientos DaliLite no es la mejor medida; y, en segundo lugar, que cada uno de los esquemas de normalización es capaz de detectar algunos tipos de plegamiento correctamente pese a fallar en otros. Por ejemplo el plegamiento IG se detecta mejor con $Norm1$ pero esta medida da el

Medida	Área	Error típ.	Intervalo de confianza asintótico al 95 %	
			Límite inferior	Límite superior
$Norm1$	0,601	0,010	0,582	0,621
$Norm2$	0,678	0,009	0,661	0,696
$Norm3$	0,637	0,009	0,619	0,656
$DaliLiteZScore$	0,772	0,009	0,755	0,789

Tabla 4.8: Valores de AUC para cada medida sobre el conjunto de datos de Fischer a nivel de clase.

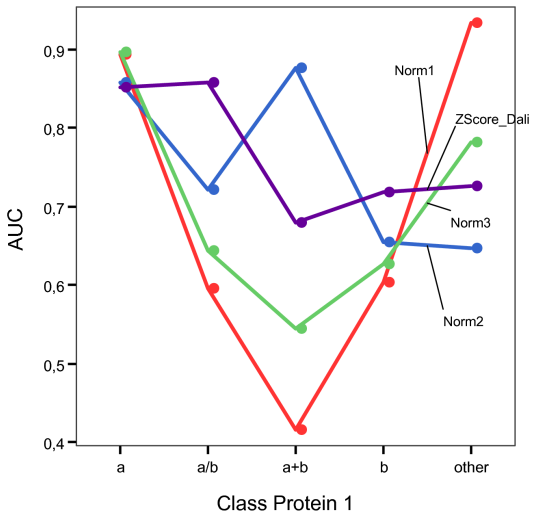


Figura 4.13: Valores de AUC para cada medida sobre el conjunto de datos de Fischer en la clasificación a nivel de la clase en SCOP, diferenciando los resultados en función de la clase de la proteína 1.

peor valor de AUC para el plegamiento TIM-barrel.

Adicionalmente, podemos analizar los resultados en términos de la clasificación a nivel de clase en SCOP, obteniéndose los valores de AUC que se muestran en la tabla 4.8. La figura 4.13 muestra una gráfica de los resultados en dicha clasificación a nivel de clase en SCOP si distinguimos los valores de AUC de cada método según cual fuera la clase de la proteína 1. Se puede ver que DaliLite obtuvo el mejor valor de AUC global pero que, si diferenciamos los resultados en función de la clase, sólo en dos de las cinco (α/β, etiquetada como a/b; y β, etiquetada como b) fue mejor. Para las otras tres clases, el valor más alto de AUC se obtiene por alguna de las normalizaciones basadas en los valores de superposición obtenidos por MSVNS2. Es también notorio que, para un total de $68 \times 68 = 4624$ pares de proteínas a comparar, DaliLite no detecta similaridad alguna para 2800 pares (60.5 %). Si se consideran los pares con Z-Score < 1 este valor aumenta hasta 3844 (83.1 %). Por el contrario, MSVNS2 es capaz de devolver diferentes grados de similaridad para todos los pares. Esta gran carencia de información, en cuanto a las similaridades no presentes, en el caso de DaliLite, es probablemente lo que hace que las normalizaciones de MSVNS2 superen a DaliLite en la detección de varias de las clases.

Experimento 3: Evaluando la similaridad en la base de datos Nh3D

El último experimento se ha realizado haciendo uso de la base de datos Nh3D v3.0 [131] de proteínas estructuralmente variadas. El conjunto de datos de Nh3D se ha construido seleccionando representantes en el nivel de topología de la base de datos CATH, eliminando dominios que contienen elementos homólogos, duplicaciones internas y regiones con alto B-Factor [131].

Nuestro objetivo fue comprobar si MSVNS2 (versión de MSVNS escogida nuevamente para obtener un buen balance entre calidad de soluciones y velocidad) produce similaridades apropiadas para clasificar las estructuras de proteínas en el nivel de arquitectura de CATH. La base de datos Nh3D contiene 806 topologías representativas que pertenecen a 40 arquitecturas. La tabla 4.9 proporciona información del tamaño de las proteínas implicadas.

La idea de este experimento es realizar la selección de un conjunto representativo de proteínas como proteínas de consulta, y compararlas contra la base de datos completa para, a continuación, usar un análisis ROC para ver

			Residuos			Contactos		
Conjunto	Proteínas	Pares	Mín.	Med.	Máx.	Mín.	Med.	Máx.
Nh3D	806	58838	33	150,33	759	74	432,83	2438

Tabla 4.9: Información del conjunto de datos Nh3D.

			Intervalo de confianza asintótico al 95 %	
Medida	Área	Error típ.	Límite inferior	Límite superior
$Norm1$	0.608	0.005	0.599	0.618
$Norm2$	0.582	0.005	0.572	0.591
$Norm3$	0.591	0.005	0.581	0.601
$DaliLiteZScore$	0.596	0.005	0.586	0.607
$ScoreMatAlign$	0.538	0.005	0.528	0.548

Tabla 4.10: Valores de AUC para cada medida sobre la base de datos Nh3D. El experimento consistió en el uso de 73 proteínas de consulta sobre 806 dominios, realizándose el análisis al nivel de arquitectura en CATH.

si los valores de similaridad obtenidos permiten discriminar qué proteinas pertenecen a la misma arquitectura de CATH. Así, para cada arquitectura (con un mínimo de 10 topologías presentes), seleccionamos la menor, la mayor y la proteína promedio en términos tanto del número de residuos como del de contactos, además de una proteína adicional seleccionada al azar. Tras eliminar duplicados obtuvimos un conjunto de 73 estructuras que son las que constituyen el conjunto utilizado como proteínas de consulta.

Cada proteína de consulta fue comparada contra cada una de las restantes proteínas en el conjunto de datos, realizando las comparaciones también con DaliLite [90] y un método recientemente propuesto de comparación de matrices de distancia, MatAlign [5], que declara ser mejor que DaliLite y CE [124] en algunos casos. Para el primero usamos el Z-Score como medida de similaridad, mientras que en el caso del segundo se usa el valor de NormScore que proporciona el método. En el caso de MSVNS2, se utilizan los 3 esquemas de normalización propuestos anteriormente en este capítulo (ecuaciones 4.1, 4.2 y 4.3). Además, el número de reinicios de MSVNS se fijó en 10 para reducir el tiempo total de ejecución dado que este conjunto de datos tiene un gran tamaño.

La figura 4.14 muestra la curva ROC para cada medida mientras que la tabla 4.10 contiene los correspondientes valores de AUC. Las medidas $Norm1$, $Norm2$ y $Norm3$ se basan en los mismos valores de superposición devueltos por nuestra estrategia, pero la normalización usada vuelve

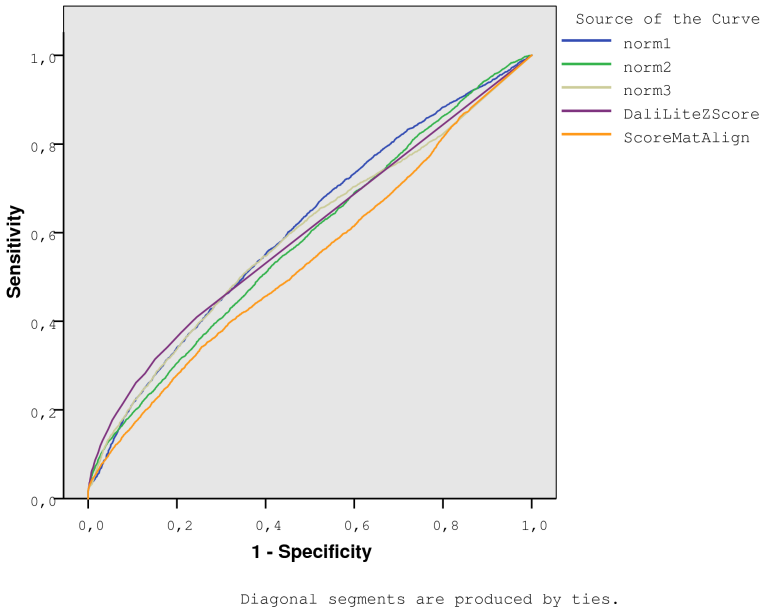


Figura 4.14: Curvas ROC para cada medida en la base de datos Nh3D para el nivel de arquitectura en CATH.

a desempeñar un papel importante. Cuando la normalización es *Norm1*, el valor de AUC es mayor que el obtenido por DaliLite, mientras que las otras alternativas de normalización obtienen valores más bajos. MatAlign obtiene claramente los peores resultados.

Si dibujamos curvas separadas, en función de la arquitectura de la proteína de consulta, se observan varios comportamientos interesantes. Algunos ejemplos de las curvas ROC para diferentes arquitecturas se muestran en la figura 4.15, observándose diferencias claras entre los métodos en función de dicha arquitectura de la proteína de consulta. Por ejemplo, $Norm1$ tiene un comportamiento excelente cuando la arquitectura de la proteína de consulta es 1.10, pero en las otras tres arquitecturas mostradas es la peor medida. De la misma manera, MatAlign obtiene resultados muy buenos cuando la arquitectura de la proteína de consulta es 3.20 y malos resultados en las otras tres. DaliLite también muestra un comportamiento diferente para las 4 arquitecturas mostradas, obteniendo buenos valores de AUC para las arquitecturas 1.20 y 3.20, mientras que en las arquitecturas 2.10 y 4.10 la curva ROC de DaliLite es prácticamente una línea diagonal muy cercana a

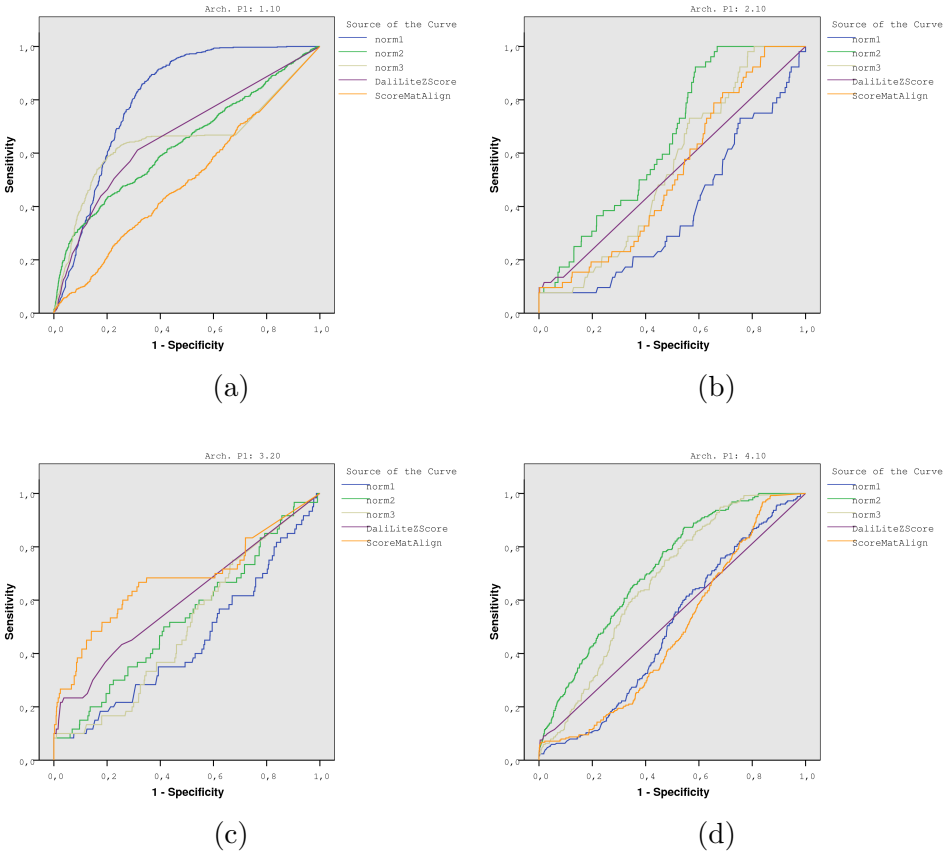


Figura 4.15: Examples of ROC curves to show how dependent the strategies are on the architecture type of the query.

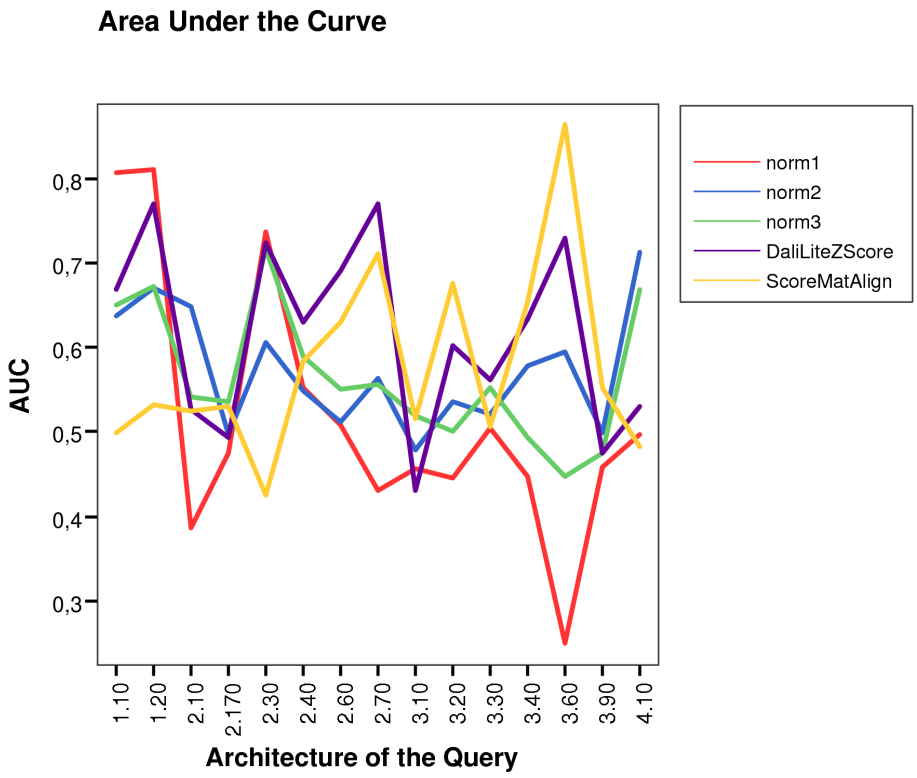


Figura 4.16: Valores de AUC para cada medida y arquitectura de la proteína de consulta en el conjunto de datos Nh3D. Se ve claramente que ninguno de los métodos se sitúa por encima de los otros para todas las arquitecturas.

la línea de no discriminación que correspondería a un clasificador aleatorio.

La figura 4.16 muestra los valores de AUC para cada arquitectura, excluyendo aquellos donde todos los métodos consiguieron AUC= 1. Si consideramos que los valores de AUC son una buena medida de la dificultad para identificar la similaridad, entonces está claro que esta dificultad difiere para cada esquema de puntuación (medida). Por ejemplo, mientras que DaliLite es un método excelente de manera global, se puede ver cómo obtiene valores de AUC muy malos cuando la arquitectura de la proteína de consulta es 3.10 y un rendimiento cercano al de un clasificador aleatorio para las arquitecturas 2.10, 2.170, 3.90 y 4.10. En el caso de $Norm1$, tenemos que se trata del mejor método de forma global, pero es obvio al observar la figura que la mayor parte de este éxito viene de un valor de AUC muy alto para proteínas

de consulta de las arquitecturas 1.10, 1.20 y 2.30, y que hay varias arquitecturas para las que obtiene los peores valores de AUC. Incluso MatAlign, que es el peor algoritmo de manera global, es capaz de obtener el valor de AUC más alto para 3 arquitecturas distintas. Estos resultados muestra que ningún algoritmo supera a los otros para todas las posibles arquitecturas de la proteína de consulta.

Debemos resaltar también que de un total de $73 \times 806 = 58838$ comparaciones de pares de proteínas, DaliLite no detectó similaridad alguna para 43833 pares (74.5 %), lo que conduce a la producción de varios falsos negativos. A modo de ejemplo, para 2 de las 7 proteínas de consulta que pertenecen a la arquitectura 4.10, DaliLite no fue capaz de retornar esas mismas proteínas como las estructuras más similares de la base de datos. Si analizamos el resultado de MSVNS sobre las mismas proteínas se obtiene que la propia proteína de consulta es retornada en los 7 casos como la más similar a sí misma.

Finalmente, para analizar los tiempos de ejecución que necesitan MatAlign, DaliLite y MSVNS hemos realizado un pequeño y simple experimento adicional. Se seleccionaron, en primer lugar, las cinco proteínas más grandes (1.10.645, 1.20.210, 3.20.70, 3.60.120 y 3.90.1300) de toda la base de datos Nh3D en base al número de contactos. Con dichas proteínas se realizó una comparación de tipo todos contra todos con objeto de filtrar los pares en los que DaliLite no detectaba suficiente similaridad como para ejecutarse. Finalmente, se ejecutaron MatAlign, DaliLite, MSVNS1, MSVNS2 y MSVNS3, en la misma máquina y bajo las mismas condiciones, para cada uno de los ocho pares no filtrados en el paso anterior. Los resultados se muestran en la tabla 4.11 y puede apreciarse claramente cómo nuestra estrategia es más rápida que DaliLite. Aunque MatAlign es aún más rápido, su mayor velocidad viene acompañada de los peores resultados en cuanto a valores de AUC, con lo que probablemente este algoritmo no esté empleando el tiempo suficiente como para conseguir un buen ranking de similaridad.

4.3. Conclusiones

En este capítulo hemos mostrado un algoritmo de Búsqueda por Entornos Variables Multiarranque (MSVNS) para el problema Max-CMO que obtiene resultados muy satisfactorios y prometedores.

Método	Tiempo(segundos)
MSVNS1	357
MSVNS2	508
MSVNS3	613
DaliLite	**758**
MatAlign	208

Tabla 4.11: Tiempos de ejecución para MSVNS, DaliLite y MatAlign. Los tiempos corresponden a la comparación de 8 pares formados por las proteínas más grandes de la base de datos Nh3D. Todas las ejecuciones se realizaron en la misma máquina y puede verse cómo MSVNS es claramente más rápido que DaliLite. MatAlign mejora los tiempos de MSVNS pero a costa de proporcionar los peores valores para realizar la clasificación.

Desde el punto de vista de la optimización, MSVNS es capaz de obtener valores de superposicion que están muy cercanos a los óptimos, usando una estrategia más simple y que requiere menos recursos computacionales que los algoritmos exactos.

Un elemento importante en varios problemas bioinformáticos es la relación entre el valor óptimo de la función objetivo y la relevancia biológica de la correspondiente solución. En la comparación de estructura de proteínas debemos recuerdar que se trabaja con modelos matemáticos que capturan algunos aspectos del problema biológico, siendo posible medir la similaridad de las estructuras de proteínas de varias maneras. Por ejemplo, hasta 37 medidas distintas fueron revisadas en [95]. Además de conseguir los valores de superposición más altos, es crítico el desarrollo de estrategias que sean capaces de obtener un buen ranking de similaridad de un conjunto dado de proteínas.

Los experimentos realizados han mostrado que, en términos del nivel de familia en SCOP y el de arquitectura en CATH, los valores de superposición (normalizados) son suficientemente buenos para capturar la similaridad. En el nivel de plegamiento en SCOP deben considerarse varios puntos. Aunque DaliLite superó a MSVNS2, los resultados de MSVNS fueron razonablemente buenos. Queda pendiente realizar más investigaciones, especialmente en el área de normalización, porque, como se vió en los experimentos, el uso de diferentes esquemas de normalización puede llevar a capacidades mejores o peores en la detección de tipos particulares de plegamientos. Debemos recordar también que todos los experimentos realizados han usado mapas de

contacto con un umbral fijo, y podría suceder que para la detección de similaridad al nivel de plegamiento se necesitara un valor diferente. Además, se mantiene abierta la cuestión de si un modelo basado en mapas de contacto puede alcanzar las prestaciones en la detección de similaridades a nivel de plegamiento que alcanza un método basado en matrices de distancia como DaliLite o no.

Los ficheros binarios ejecutables para las plataformas Windows y Linux y las instrucciones de uso de MSVNS están disponibles en http://modo.ugr.es/jrgonzalez/msvns4maxcmo. Adicionalmente, el método fue aceptado para su incorporación en el servidor ProCKSI [8]. ProCKSI corresponde a las iniciales en inglés de "Protein Comparison, Knowledge, Similarity and Information" (Comparación de proteínas, conocimiento, similaridad e información). Es un servidor multicapa de comparación de proteínas que calcula las similaridades estructurales usando medidas de teoría de la información (USM). ProCKSI se enlaza a una buena variedad de fuentes externas y va incorporando paulatinamente métodos adicionales para mejorar e incrementar sus hallazgos en materia de similaridad. Nuestro algoritmo MSVNS fue escogido como método para resolver el modelo Max-CMO en ProCKSI debido a la calidad de sus soluciones y su velocidad, factores que son primordiales para un servidor de acceso público en la red como es ProCKSI.

Capítulo 5

Comparación de proteínas con mapas de contacto difusos

En el capítulo anterior se ha detallado la investigación realizada para la comparación de estructuras de proteínas usando mapas de contacto estándar. En este capítulo se extiende dicha investigación al incorporar los mapas de contacto de tipo difuso y estudiar su resolución tanto mediante el uso de la medida universal de similaridad (USM), como de una nueva versión del algoritmo VNS, adaptada para trabajar con el nuevo tipo de mapas de contacto haciendo uso del modelo generalizado de la máxima superposición de mapas de contacto difusos (GMax-FCMO).

5.1. Comparación de proteínas mediante mapas de contacto difusos y la medida universal de similaridad

En el capítulo de Bioinformática se presentó la medida universal de similaridad (USM), que ya había sido utilizada como medida de similaridad en la comparación de proteínas usando mapas de contacto estándar [77]. Sin embargo, la definición de mapas de contacto difusos supone una nueva fuente de información de una proteína sobre la que se puede aplicar la medida USM. En el citado capítulo 2 se puede encontrar una explicación detallada de USM y de los mapas de contacto, con lo que aquí simplemente haremos,

a modo de recordatorio, un breve resumen:

Los mapas de contacto difusos son representaciones de una proteína que hacen uso de las distancias entre los residuos para establecer si existe o no contacto entre ellos, permitiéndose, además, la existencia de varios tipos de contacto difusos con distintas funciones de pertenencia, lo cual es su principal diferencia con los mapas de contacto estándar. Además, dado que cada tipo de contacto lleva asociada una función de pertenencia, cada contacto tendrá asociado un determinado grado de pertenencia.

Por otro lado, USM usa el concepto de complejidad de Kolmogorov para calcular la similaridad entre dos objetos cualesquiera en base a una ecuación (que recordaremos posteriormente) cuyos términos pueden aproximarse mediante el cálculo de los tamaños de compresión de ficheros cuyo contenido sea una representación adecuada de los objetos a comparar. En concreto, los mapas de contacto pueden representarse de forma matricial y dicha representación es susceptible de ser almacenada de forma textual en un fichero que serviría como punto de partida para la aplicación de USM.

5.1.1. Experimentos

Para comprobar los beneficios de utilizar mapas de contacto de tipo difuso, junto a la medida universal de similaridad, como instrumento para la comparación de estructuras de proteínas, tomamos el conjunto de datos de Chew-Kedem [24], compuesto de 32 proteínas de tamaño medio agrupadas en cinco familias diferentes: globins (1eca, 5mbn, 1hlb, 1hlm, 1babA, 1ithA, 1mba, 2hbg, 2lhb, 3sdhA, 1ash, 1flp, 1myt, 1lh2, 2vhb), alpha-beta (1aa9, 1gnp, 6q21, 1ct9, 1qra, 5p21), tim-barrels (6xia, 2mnr, 1chr, 4enl), all beta (1cd8, 1ci5, 1qa9, 1cdb, 1neu, 1qfo) y alpha (1cnp, 1jhg). Todas estas proteínas fueron obtenidas a través de la base de datos del Protein Data Bank [14], y se hace uso de este conjunto de datos con propósitos comparativos, por ser el mismo que ya se había utilizado para aplicar USM con mapas de contacto estándar en [77].

A continuación construimos 6 definiciones diferentes para los mapas de contacto difusos (ver figura 5.1), que intentan representar un buen abanico de opciones desde una definición equivalente a la de un mapa de contacto estándar (a) hasta el uso de 3 tipos de contacto difusos (d), pasando por la generalización difusa más simple (b) y diferentes definiciones con dos tipos de contactos (c, e y f). Con esto se pretende averiguar cuánta información

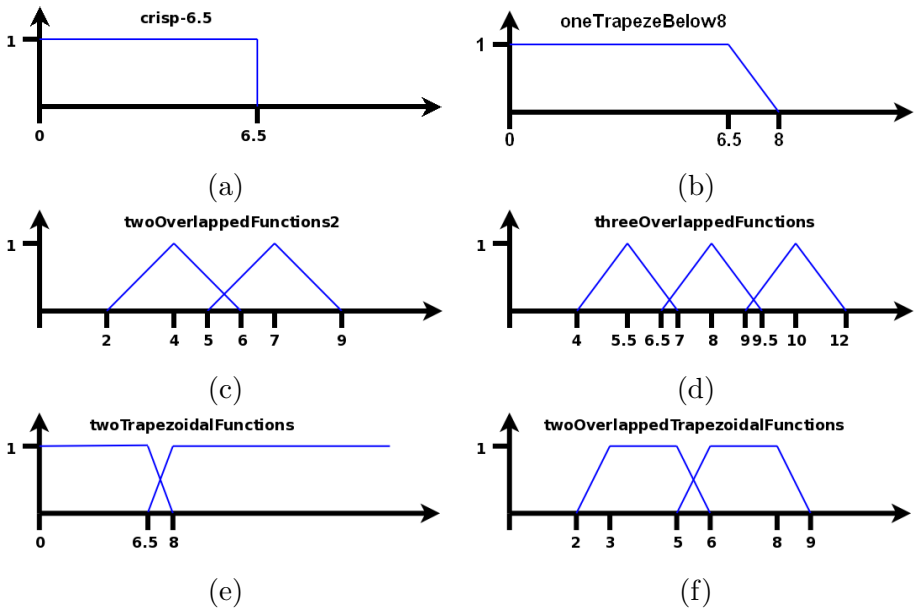


Figura 5.1: Definiciones de funciones de pertenencia usadas para crear los mapas de contacto difusos para USM.

debería tener un mapa de contacto para proporcionar buenos resultados, y para ello se siguen los siguientes pasos:

- Para cada fichero .pdb de las proteínas del conjunto de datos se extrajo la primera cadena.
- Para cada cadena y cada definición de funciones de pertenencia (figura 5.1) se produjo un fichero de mapa de contacto difuso. Dicho fichero se construyó usando una representación textual de la matriz del mapa de contacto difuso, colocando los tipos de los contactos en la matriz triangular inferior y los grados de pertenencia en la matriz triangular superior.
- Para cada par de ficheros c_1, c_2 de mapas de contacto difusos de las proteínas (con las mismas funciones de pertenencia), se calcula $d(c_1, c_2)$ haciendo uso de la ecuación 5.1 para obtener la similaridad entre ambos, almacenando dichas similaridades en una matriz. Los valores de cualquiera de los $K(o)$ en la ecuación 5.1 son aproximados mediante el tamaño del fichero o comprimido, mientras que el cálculo de $K(o_1|o_2)$ se hace mediante $K(o_1 \cdot o_2) - K(o_2)$ donde $\cdot$ denota concatenación de

ficheros.

$$d(o_1, o_2) = \frac{\text{máx}\{K(o_1|o_2^*), K(o_2|o_1^*)\}}{\text{máx}\{K(o_1), K(o_2)\}} \quad (5.1)$$

- Aplicar agrupamiento por enlace simple sobre cada una de las 6 matrices obtenidas en los pasos anteriores (una para cada definición diferente de funciones de pertenencia para los mapas de contacto difusos).

Los resultados obtenidos usando este protocolo se muestran como dendogramas en la figura 5.2. Cada dendograma corresponde a la representación gráfica del resultado del agrupamiento por enlace simple para una de las 6 matrices de resultados. El caracter entre paréntesis corresponde a las definiciones mostradas en la figura 5.1, de manera que el dendograma (a) se corresponde al cluster obtenido usando mapas de contacto estándar (crisp). Para simplificar el análisis, se ha añadido un símbolo a la derecha del nombre de la proteína, que denota la clase a la que dicha proteína pertenece.

Es interesante observar que en todos los cluster aparecen una serie de características en común. En primer lugar parece fácil la realización de un buen agrupamiento de algunas clases, concretamente esto ocurre para las clases globins (marcada con $\sim$), alpha ($+$) y tim-barrels ($\#$). La clase "all beta" ($*$) es la clase más difícil, ya que tiene seis proteínas y aparece separada en dos grupos en la mitad de los dendogramas (en concreto, los dendogramas a, c y d), con 3 proteínas en cada uno de estos dos grupos. Otra característica en común es que la proteína 1ct9, de la clase alpha-beta ($\wedge$), siempre aparece agrupada incorrectamente con el grupo de las proteínas tim-barrel ($\#$).

La generalización más simple sobre la definición estándar de mapa de contacto es la correspondiente a la definición (y al dendongrama) etiquetada como (b), y permite obtener una detección casi perfecta de las clases. Solamente la proteína 1ct9 se encuentra mal clasificada, lo cual no se conseguía con el modelo estándar de mapas de contacto, y supone por tanto una mejora.

El dendograma (e), construido a partir de mapas de contacto que incluyen la definición usada en (b), también alcanza un cluster casi perfecto pero con un orden diferente al de (b). No obstante, los mapas de contacto generados por la definición (e) tienen más información que los de (b), al incorporar la misma función de pertenencia de "contacto corto" y extenderla

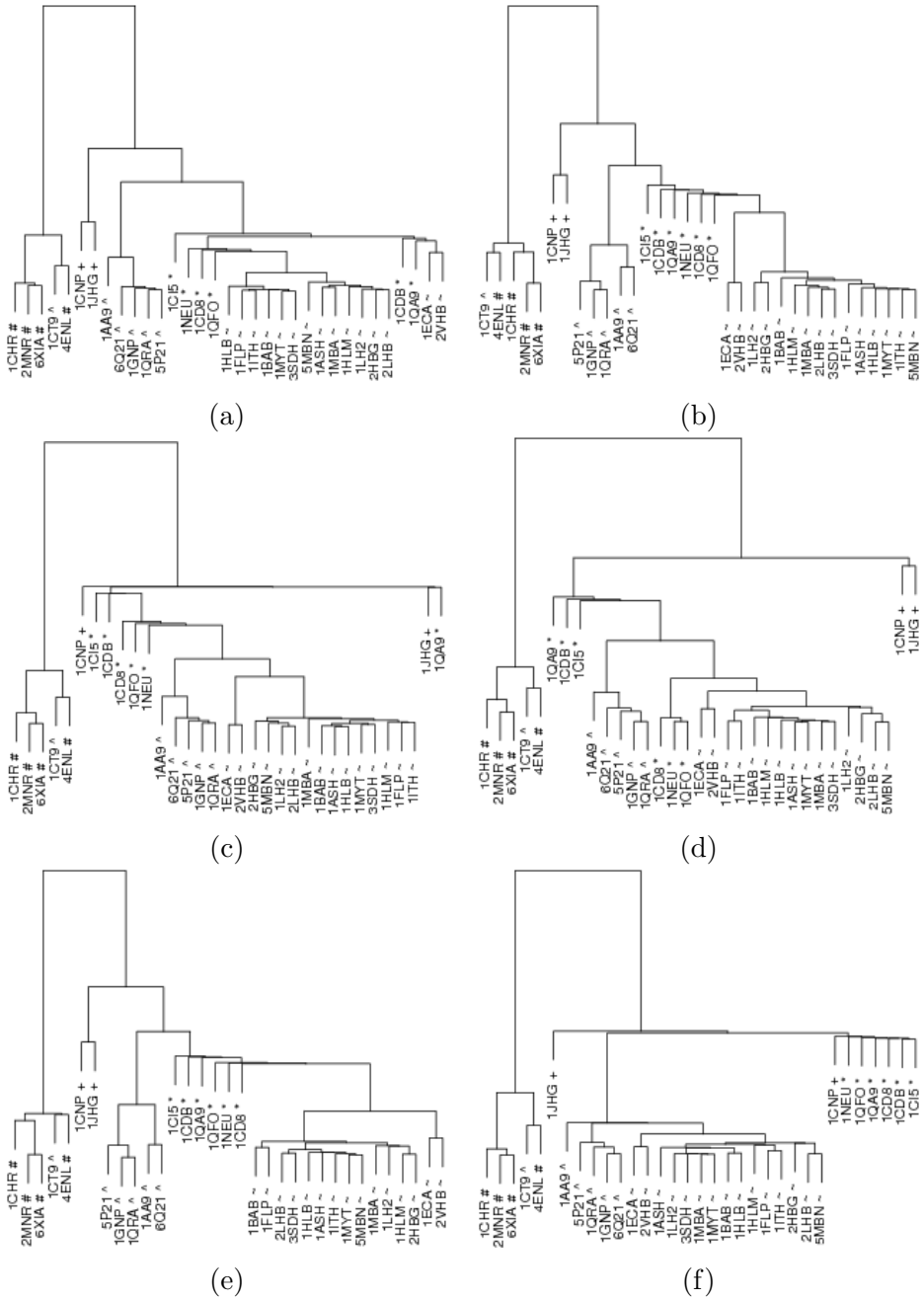


Figura 5.2: Clusters obtenidos para las matrices de similaridad resultantes de la comparación mediante USM de los mapas de contacto difusos de las proteínas de Chew-Kedem. Las letras entre paréntesis se corresponden con las 6 definiciones de funciones de pertenencia usadas para crear los mapas de contacto difusos, que se mostraban en la figura 5.1.

con una función de pertenencia para "ausencia de contacto". La matriz de los mapas de contacto para (e) almacena pues información de cada distancia presente, de manera que no hay ninguna entrada con valor cero, ni en los grados de pertenencia ni en los tipos de contacto. La mayoría de los valores son 1, y posiblemente esta información nueva puede comprimirse extraordinariamente bien, de manera que no deteriora los resultados como ocurre en (d). En otras palabras, la información que se gana con la parte más a la derecha en la definición (e) es innecesaria.

El dendograma (f) también es interesante porque muestra una perfecta detección de las clases $\#, \wedge$ y $*$. En cambio, las dos proteínas de la clase $+$ aparecen separadas. Una de ellas aparece sola en una rama del árbol mientras que la otra se empareja con las de la clase $*$.

Las definiciones (c) y (f) difieren en el tipo de funciones de pertenencia usadas, aunque cubren la misma región de distancias. Los mapas de contactos correspondientes a ambas funciones son iguales en términos del tipo de los contactos, pero difieren en los grados de pertenencia. Dichos grados de pertenencia son mayores cuando se aplica la definición (f). Este cambio en los mapas de contacto difusos permitió obtener mejores resultados con respecto a (c). Esto es así porque con (c) tanto las clases $+$ como $*$ aparecen separadas, mientras que en (f) solamente la clase $+$ es la que se divide inapropiadamente. Esto nos sugiere que es mejor hacer uso de funciones de pertenencia de tipo trapezoidal frente a las de tipo triangular.

En resumen, si no consideramos la proteína 1CT9, las definiciones (a) y (c) condujeron a dendogramas donde había dos clases que aparecían separadas; las definiciones (d) y (f) solamente dividieron una clase; y las definiciones (b) y (e) condujeron a la obtención de dendogramas con una recuperación perfecta de todas las clases.

Los resultados de este experimento fueron presentados en los congresos EUSFLAT-LFA 2005 [51] y EURO XXI 2006 [55].

5.2. Resolución del GMax-FCMO

Para la resolución del problema GMax-FCMO (ver sección 2.2.9) se ha realizado una adaptación del algoritmo MSVNS presentado en el capítulo anterior. El algoritmo sigue exactamente el mismo esquema presentado en el capítulo anterior y simplemente se ha adaptado el código para permitir

la lectura de ficheros de mapas de contacto difusos y para que realice el cálculo de los valores objetivos de las soluciones según las fórmulas propias del modelo GMax-FCMO. Esta versión de MSVNS, junto a los experimentos que se presentan en las siguientes subsecciones, han sido presentadas en el congreso FUZZ-IEEE 2007 [56].

De cara a testear el desempeño del MSVNS en este nuevo modelo, se han desarrollado una serie de experimentos. El objetivo de estos experimentos es, en primer lugar, comprobar si los valores de similaridad (en términos de GMax-FCMO) entre las proteínas de consulta y las proteínas en la base de datos son mayores para las proteínas de la misma clase. Asimismo, se quiere analizar la relación entre los valores de GMax-FCMO y Max-CMO, para ver si el uso de tipos de contactos tiene sentido o no. Las siguientes secciones describen los experimentos realizados.

5.2.1. Experimento 1: ¿Puede detectarse la similaridad entre proteínas usando GMax-FCMO?

Para responder a la pregunta que marca el objetivo de este experimento se ha usado una versión reducida del conjunto de datos de Chew-Kedem [24] compuesta por 25 proteínas que pertenecen a cinco arquitecturas de proteínas diferentes (según el criterio de la base de datos SCOP [2, 105]):

1. *Clase alpha (a)*: 1cnpA, 1jhgA.
2. *Clase alpha-beta (ab)*: 1aa9, 1gnp, 1qra, 5p21, 6q21.
3. *Clase all beta (b)* : (1cd8, 1cdb, 1ci5, 1hnf, 1neu, 1qa9, 1qfo).
4. *Clase globins*: 1ash, 1babB, 1flp, 1hlm, 1myt, 2lhb, 3sdhA.
5. *Clase tim-barrels*: 1chr, 2mnr, 4enl, 6xia.

En relación al conjunto de datos de Chew-Kedem original, nuestro conjunto de prueba incluye todas las proteínas de las clases *a*, *ab*, *b* y *tim-barrel* y solamente el 50 % de las que pertenecen a la clase *globins*, para mantener un número similar de proteínas en cada clase.

A continuación, en lugar de realizar comparaciones entre todos los pares posibles (tal y como se hizo en el USM), vamos a realizar consultas (queries) sobre ella con proteínas de clase conocida. Además dado que la experiencia

del caso del USM aconsejaba usar funciones de pertenencia de tipo trapezoidal y no usar un número excesivo de tipos de contacto (que conducen a la recopilación de información innecesaria) el procedimiento de comparar las proteínas de consulta con las de la base de datos se ha realizado únicamente para tres definiciones alternativas de mapas de contactos (ver figura 5.3) que se explican más adelante.

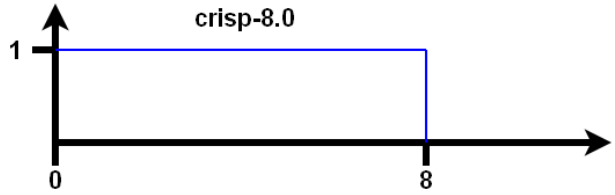
La tabla 5.1 contiene un sumario de las clases del conjunto de datos, en términos de los tamaños de proteínas correspondientes. El número de contactos se ha calculado para un mapa de contacto estándar con un umbral de 8Å. Puede observarse que las variaciones dentro de una misma clase son pequeñas, tanto en términos del número de contactos como en el de residuos, excepto en la clase *beta*, donde los valores máximos prácticamente doblan a los mínimos. Las diferencias entre clases son bastante mayores, con las proteínas más grandes en la clase de las tim-barrels y las más pequeñas en la clase alpha.

	Residuos				Contactos a 8Å			
Clase	Media	Desv.est.	Mín	Máx	Media	Desv.est.	Mín	Máx
a	95.50	5.503	90	101	296.00	7.004	289	303
ab	168.00	2.450	166	171	652.80	33.079	618	713
b	117.86	25.940	95	179	429.71	99.845	364	668
globins	147.57	4.687	142	158	519.14	13.370	497	532
$TIMBarrel$	387.50	29.962	357	436	1627.00	174.572	1472	1914

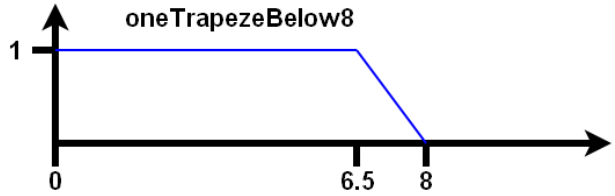
Tabla 5.1: Análisis de los tamaños de las proteínas en cada clase del conjunto de datos.

Para verificar que los valores de similaridad obtenidos por el MSVNS tienen sentido y que tener mapas de contacto difusos es útil, se procedió de la siguiente forma:

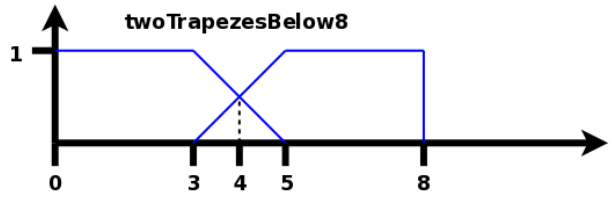
- Para cada clase, la primera proteína se comó como proteína representativa de la misma: 1aa9 para alpha-beta, 1cnpA para alpha, 1cd8 para all beta, 1ash para globins y 1chr para tim-barrels.
- Se construyeron tres definiciones distintas de mapas de contacto, tal y como se ve en la figura 5.3, todas ellas cubriendo el mismo rango de distancias entre residuos (desde 0 hasta 8Å). De esta forma se llega a mapas de contacto que tienen el mismo número de contactos, pero



(a)



(b)



(c)

Figura 5.3: Definiciones usadas para crear los mapas de contacto difusos para MSVNS.

mientras (a) es una función puramente dicotómica que hace que el modelo GMax-FCMO sea equivalente al Max-CMO estándar con un umbral de 8Å, tanto (b) como (c) introducen las alternativas difusas en el modelo. Así, la definición (b) contiene una única función que es exactamente igual a la función estándar, pero con una pendiente decreciente que reduce el grado de pertenencia de un contacto a medida que la distancia se acerca desde 6.5 hasta 8Å. La definición (c), en cambio, divide el rango de distancias en dos regiones superpuestas, conduciendo a mapas de contacto donde el "tipo" de contacto pasa a desempeñar un papel.

- Para cada una de las definiciones, y cada una de las proteínas representativas (proteínas de consulta), se realiza una comparación a nivel de pares con todas las restantes proteínas del conjunto de datos (incluyendo la comparación con sí mismas).

El resultado esperado es que los valores más altos de similaridad de la proteína de consulta con las restantes sean mayores para aquellas proteínas que pertenecen a su misma clase. Pero dado que el valor objetivo depende del tamaño de las proteínas y del número de contactos, se necesita normalizar los valores obtenidos con el objetivo de poder compararlos. La normalización se realiza según la siguiente fórmula:

$$normFitness(P_i, P_j) = \frac{fitness(P_i, P_j)}{max\{fitness(P_i, P_i), fitness(P_j, P_j)\}}$$

donde el término $fitness(P_k, P_k)$ es una medida de "auto-similaridad", calculada como el valor de la superposición de una proteína consigo misma. No se usan aquí las normalizaciones $Norm1$, $Norm2$ y $Norm3$, propuestas en el capítulo anterior, porque dichas normalizaciones están pensadas exclusivamente para el modelo Max-CMO, donde el número de contactos y el valor de autosimilaridad coinciden. Esto no sucede así con el modelo GMax-FCMO y su diferente función de costo, con lo que, para conseguir similaridades normalizadas en el intervalo $[0, 1]$, se hace necesario introducir la fórmula anterior.

La figura 5.4 muestra los valores medios de *normFitness* para cada combinación de la clase de la proteína de consulta (classP1) contra cada una de las clases del conjunto de datos (classP2) y cada definición de mapa de contacto. Estos valores pueden entenderse como las similaridades entre la

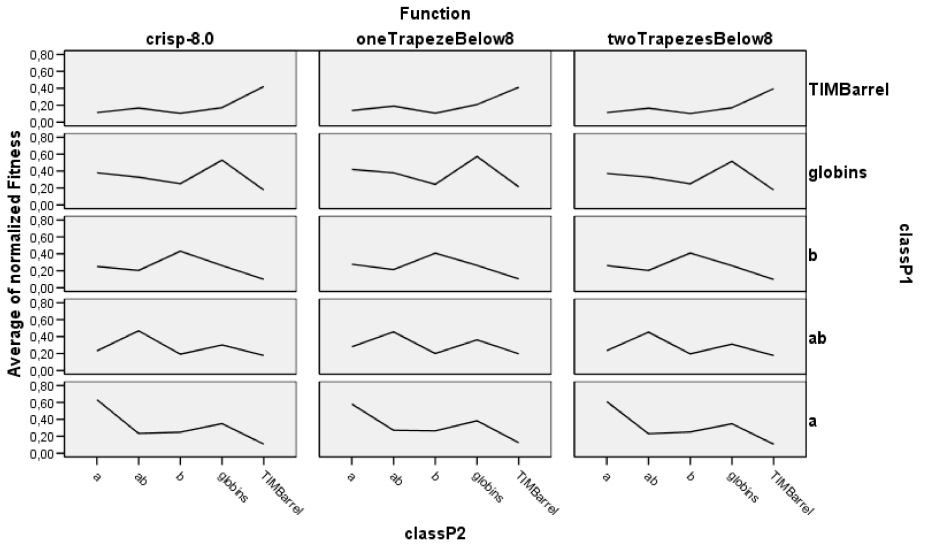


Figura 5.4: *normFitness* medio para cada par de clases con cada definición (función).

proteína de consulta y las correspondientes clases. Se puede observar fácilmente que para cada clase de la proteína de consulta (classP1 que se muestra en las filas), todos las definiciones de mapas de contacto conducen a valores mayores de *normFitness* para el grupo de proteínas de su misma clase (classP2 que se muestra en las columnas), de tal manera que cada curva de valores objetivo (fitness) alcanza su máximo para los puntos donde classP1 es igual a classP2. Por ejemplo, cuando se hace una consulta con una proteína de la clase alpha (indicada como "a" en la fila de debajo), la similaridad media más alta es con las proteínas de esa misma clase "a", y esto se cumple para cada clase y para cada una de las 3 representaciones de mapas de contacto utilizadas.

También debe indicarse que el comportamiento anterior no se cumple únicamente en términos del *normFitness* medio, sino que cualquiera que sea la proteína de consulta escogida, todas las proteínas de su misma clase obtienen mayores valores de fitness y *normFitness* que cualquier otra proteína del conjunto de datos. Esto confirma que, en este conjunto de datos, los valores normalizados son suficientemente buenos para distinguir proteínas de las mismas clases tanto con los mapas de contacto generados mediante

tipos de contacto estándar (función de pertenencia con valores discretos 0 ó 1) como con los tipos de contacto difusos (funciones de pertenencia con grado de pertenencia variando entre 0 y 1) e independientemente de si se hace uso o no de tipos de contactos.

Por tanto, los resultados obtenidos se ajustan a lo esperado, obteniéndose medidas de similaridad con la resolución del modelo GMax-FCMO que son lo suficientemente buenas como para distinguir las proteínas de la misma clase. Este resultado complementa el obtenido en la sección 5.1 donde en lugar del modelo GMax-FCMO se usaba el USM para la obtención de las similaridades entre los mapas de contacto difusos.

5.2.2. Experimento 2: Relación entre los valores de superposición estándar y difusos

Si recordamos las definiciones de los problemas Max-CMO y GMax-FCMO del capítulo 2, podríamos asumir que los valores de fitness obtenidos por la función crisp-8.0 son una cota superior para los valores de fitness que se pueden alcanzar con las definiciones difusas oneTrapezeBelow8 y twoTrapezesBelow8. Esto es así porque, en el Max-CMO, cada ciclo de longitud cuatro incrementa el valor de la superposición en una unidad, mientras que, en el caso difuso, la contribución de cada ciclo puede variar en el rango [-1, 1].

Considerando las diferentes escalas en que se mueven los valores de superposición, la comparación entre valores procedentes de diferentes definiciones de mapas no es fácil. Sin embargo, se ha diseñado un procedimiento sencillo que permite realizar algunos análisis adicionales, que consiste en dada una solución concreta (un alineamiento) que se ha obtenido mediante la resolución de una instancia del problema GMax-FCMO, calcular el valor de superposición equivalente al alineamiento si los mapas de contacto fueran estándar. De esta forma se puede comparar la solución final obtenida con oneTrapezeBelow8 o twoTrapezesBelow8 con la función crisp-8.0 no difusa.

Tras ello, realizamos un análisis ANOVA y un test de Tahmane post-hoc para comprobar si las diferencias entre los valores medios obtenidos con cada una de las tres definiciones de mapas de contacto son significativas o no. Los resultados indican que no hay diferencia estadísticamente significativa entre oneTrapezeBelow8 y twoTrapezesBelow8, y que, aunque crisp-8.0 es estadísticamente mejor que oneTrapezeBelow8, este no es el caso cuando se

compara con twoTrapezesBelow8, en cuyo caso no existen diferencias claras. En otras palabras, si no se usa el tipo de contacto, el modelo estándar es mejor que la más simple de las generalizaciones difusas (crisp-8.0 frente a oneTrapezeBelow8); pero si el tipo de contacto entra en juego (twoTrapezesBelow8), la calidad de las soluciones obtenidas con el modelo difuso pasa a ser indistinguible (estadísticamente hablando) de la calidad de las soluciones del modelo estándar. Este es otro factor importante que nos permite decir que la adición de cierto significado semántico de los contactos tiene sentido, tal y como se vió al aplicar USM (sección 5.1).

Estudiando en mayor detalle los resultados procedentes de la definición difusa twoTrapezesBelow8, expresados en términos del valor equivalente en Max-CMO, se puede observar que, en varias ocasiones, dichos valores son mayores que los obtenidos de resolver el modelo no difuso (definición crisp-8.0) directamente. Este comportamiento tiene una implicación importante: *se puede resolver el problema Max-CMO mediante la resolución del problema GMax-FCMO.*

Conviene notar que si la resolución de un problema de Max-CMO (como el que se deriva del uso de la función crisp-8.0) se realizara hasta la optimalidad, el valor de superposición obtenido tendría que ser siempre mayor que el valor obtenido de los mapas difusos. Pero como el problema se resuelve generalmente de forma heurística (debido a su naturaleza NP-completa), esta relación no tiene por qué mantenerse necesariamente. De hecho, como hemos visto antes, en nuestro experimento obtuvimos, para algunas instancias, valores de superposición mayores, en el sentido de Max-CMO, al resolverlas mediante el GMax-FCMO y los mapas de contacto difusos.

La cuestión que queda por ver ahora es en qué instancias del problema se produce esta situación y qué características tienen dichas instancias. Para intentar responder a esta cuestión, primero consideramos el número de contactos de las proteínas que eran comparadas como una posible causa. De este modo, para cada par de proteínas, tomamos el máximo valor de superposición obtenido tanto por los mapas procedentes de crisp-8.0 como los de twoTrapezesBelow8 y calculamos la diferencia. Del mismo modo, calculamos la diferencia en número de contactos y residuos de las proteínas que son objeto de comparación.

El diagrama de puntos (scatter plot) de estos valores para cada par de proteínas se muestra en las figuras 5.5 y 5.6. Un punto en el gráfico es positivo

(y mostrado como un círculo) cuando el valor de superposición (fitness) correspondiente a mapas de crisp-8.0 es mejor, y negativo (mostrado como un rombo) cuando el fitness (no difuso) de los mapas de twoTrapezesBelow8 es mejor, estando reflejadas las diferencias en coste mediante una escala de tipo logarítmico.

Dos grupos aparecen claramente si consideramos las diferencias en el número de contactos alrededor del valor 500. Los pares donde la resolución a través de los mapas difusos ayuda más a mejorar el fitness son principalmente los pares donde las diferencias en número de contactos de las dos proteínas no son demasiado grandes (menos de 500 contactos de diferencia). De manera más precisa, el uso de twoTrapezesBelow8 es mejor que el uso de crisp-8.0 en 29 de 83 casos (35 %) para pares de proteínas con una diferencia de menos de 500 contactos, y solamente en 7 de 37 casos (19 %) para pares con una diferencia de más de 500 contactos. Pero hay que notar que estas diferencias de contactos más grandes sólo son posibles cuando una de las proteínas comparadas pertenece a la clase tim-barrel (que son las mayores tanto en término del número de residuos como en el de contactos, como puede verse en la tabla 5.1). Se obtiene un resultado similar cuando se consideran las diferencias en términos del número de residuos, donde también los mapas de contacto difusos alcanzan frecuentemente mejores resultados que los de crisp-8.0 cuando la diferencia no es demasiado grande. Esto se debe probablemente a una alta correlación entre el número de contactos y el número de residuos, con lo que cualquiera de estas dos propiedades es equivalente en cuanto al comportamiento asociado al uso de mapas de contacto difusos que se puede esperar.

Por otro lado, como las proteínas de la clase tim-barrel son las proteínas implicadas en las mayores diferencias de contactos/residuos, pueden estar influyendo también en los resultados algunas características propias de esta clase. En el ámbito de esta tesis nos quedamos con que el uso de la diferencia de contactos o residuos (de manera indistinta) es uno de los factores que pueden aconsejar el uso de GMax-FCMO frente al de Max-CMO, pero no ha sido posible explorar este tema en más detalle. Consideramos que queda abierta la cuestión de identificar más características relevantes de las proteínas (como puede estar siendo el caso de la clase tim-barrels) que sirvan de indicadores para saber cuándo es interesante calcular el valor de Max-CMO

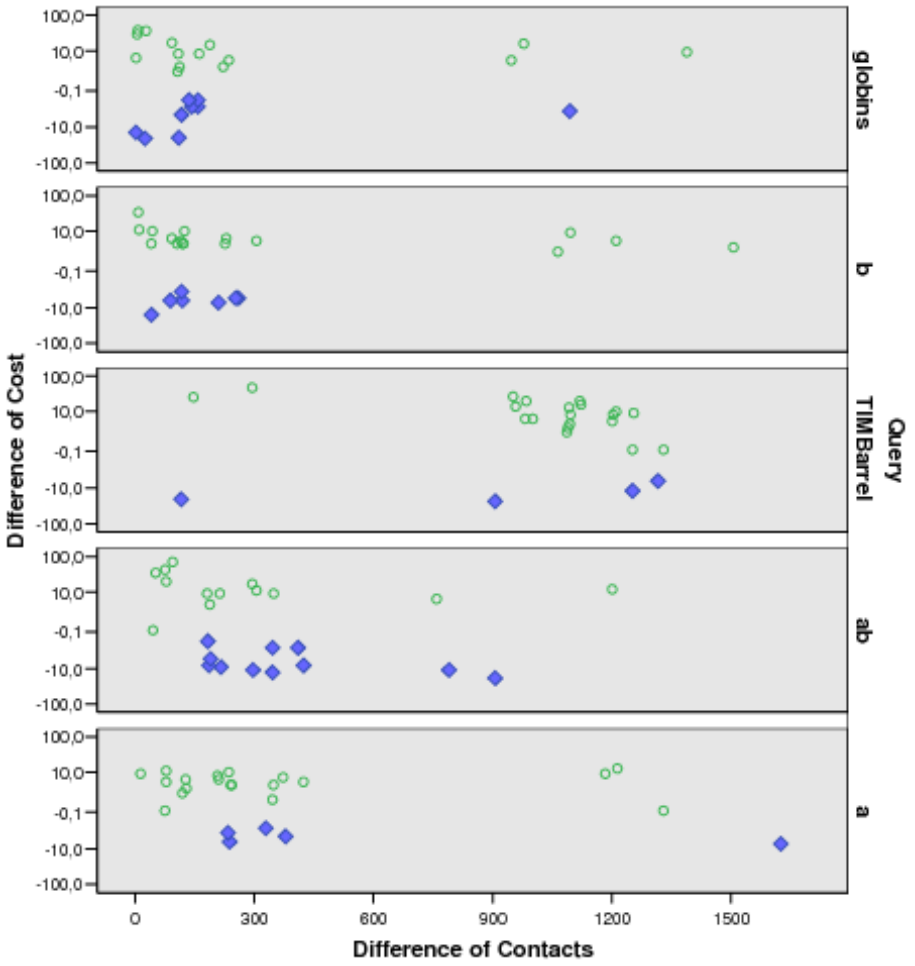


Figura 5.5: Dispersión de las diferencias de costo entre crisp-8.0 y twoTrapezesBelow8 frente a la diferencia de contactos.

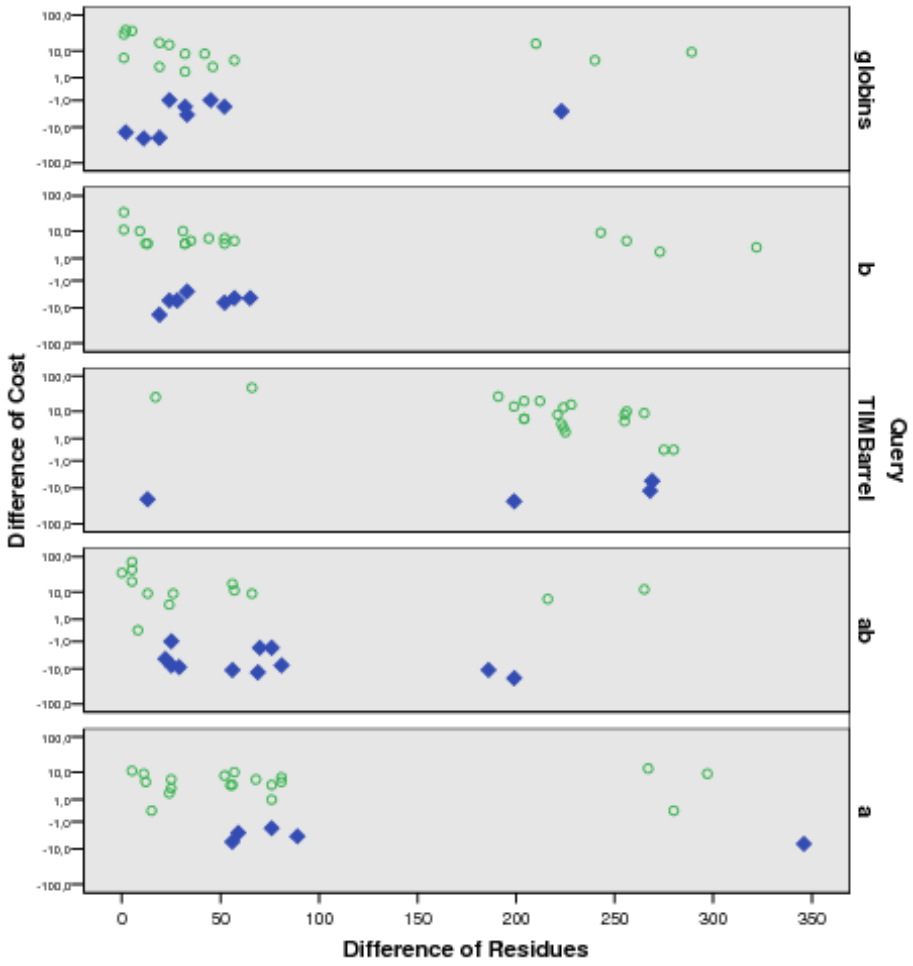


Figura 5.6: Dispersión de las diferencias de costo entre crisp-8.0 y twoTrapezesBelow8 frente a la diferencia de residuos.

a través de mapas de contacto difusos y cuándo no.

5.3. Conclusiones

En este capítulo hemos mostrado dos métodos de comparación de proteínas utilizando mapas de contacto de tipo difusos, el primero de los cuales se basa en la medida universal de similaridad (USM) y sirve como extensión a los trabajos previos donde se realizaba la comparación mediante USM de mapas de contacto estándar; mientras que el segundo utiliza una adaptación del algoritmo MSVNS visto en el capítulo anterior, que ha sido adaptada para trabajar con mapas de contacto de tipo difuso y el modelo GMax-FCMO de comparación de los mismos.

En primer lugar, se ha podido observar cómo la utilización de mapas de contacto difusos, en lugar de los estándar, tiene sentido y puede dar lugar a mejores resultados incluso en términos de valores de Max-CMO. Hemos visto que, aunque no resulta beneficioso utilizar un número excesivo de funciones de pertenencia o tipos de contactos, las generalizaciones más simples, con una o dos funciones de pertenencia para tipos de contacto difusos, pueden obtener mejores resultados que los mapas de contacto estándar. Esto sucede tanto a la hora de recuperar mejor la jerarquía de las clases usando clustering jerárquico (sección 5.1.1) como en la correcta distribución de los valores de similaridad (sección 5.2.1) de forma que las proteínas de las mismas clases obtienen mayores valores de similaridad que las que son de clases distintas. Por último, se ha podido observar (sección 5.2.2) que la comparación con mapas de contacto difusos puede ser beneficiosa, incluso, para obtener en algunos casos mejores valores de Max-CMO de los que se consiguen al comparar los mapas de contacto estándar directamente, lo que seguramente se deba a que el espacio de búsqueda que define el modelo GMax-FCMO, con una función de costo más continua, puede llegar a ser más adecuado para los algoritmos heurísticos que el asociado a Max-CMO con su función de costo discreta que solamente suma el número de ciclos de cada solución.

Conclusiones

La investigación realizada en esta tesis ha versado sobre la aplicación de técnicas basadas en Soft Computing y SAD al problema de comparación de estructuras de proteínas. Los objetivos concretos que se perseguían eran los siguientes:

1. Proporcionar métodos eficientes y de calidad para la comparación de estructuras de proteínas, que complementen y/o mejoren los ya existentes y posibiliten el mejor desarrollo de otras investigaciones.

2. Utilizar la Soft Computing para la creación de los métodos anteriores, de forma que se puedan aprovechar sus cualidades para la creación de métodos mejores.

3. Facilitar herramientas y técnicas que permitan tratar con la información imprecisa propia del problema de comparación de proteínas y, en particular, profundizar en el uso de una modelización difusa del problema de comparación.

4. Comprobar que los métodos anteriores tienen utilidad desde el punto de vista biológico.

5. Crear un SAD que ayude en la experimentación asociada a los nuevos métodos, permitiendo manejar de forma sencilla todos los métodos y datos asociados al problema de comparación de proteínas.

En lo que al desarrollo de SAD se refiere (objetivo 5), se ha conseguido completar con éxito el sistema SiGMA, que va más allá de las pretensiones iniciales de esta tesis. En efecto, SiGMA es un esqueleto para el desarrollo de SAD basados en optimización, que se enfoca al manejo y ejecución de modo dinámico de los *resolutores* y al análisis de sus resultados. SiGMA fue

diseñado teniendo en mente la experimentación específica necesaria para la tesis, pero se ha mostrado cómo, en su versión genérica (con la funcionalidad básica del esqueleto), puede cumplir toda la funcionalidad deseada para esta clase de SAD, tal y como se detalló en el capítulo 3. De esta forma, SiGMA es una herramienta valiosa para el trabajo con cualquier tipo de *resolutores* (algoritmos) de modo independiente de los problemas/modelos que resuelven. Se ha mostrado también cómo se puede usar SiGMA para construir SAD basados en optimización más complejos, que proporcionan funcionalidades más avanzadas, cuando se tiene conocimiento específico del problema, dando dos ejemplos: SiGMAPhub, para el problema del p-hub mediano, y SiGMAProt, para la comparación de estructuras de proteínas. En concreto, SiGMAProt ha cumplido los objetivos para los que fue creado, al permitirnos realizar de forma efectiva toda la experimentación propia de la tesis y el manejo de los resultados obtenidos.

Por otro lado, en cuanto al desarrollo de métodos de comparación de proteínas basados en Soft Computing (objetivos 1-4), la tesis presenta resultados importantes en el área de comparacion de proteínas a través de mapas de contacto estándar y difusos. Para ello se desarrolló un algoritmo multiarranque de búsqueda por entornos variables (MSVNS) que permite resolver el modelo de máxima superposición de mapas de contacto tanto en su formulación estándar (Max-CMO) como en la formulación generalizada difusa (GMax-FCMO). Hemos mostrado cómo MSVNS es un algoritmo importante y útil en varios aspectos. En primer lugar, porque consigue valores de superposición óptimos o cercanos al óptimo usando una estrategia más sencilla y menores recursos computacionales. De esta forma, MSVNS se ha convertido en la mejor heurística que existe en la actualidad para resolver este modelo de comparación, cumpliéndose con ello los objetivos 1 y 2. Además, la aplicación de técnicas de agrupamiento sobre los resultados proporcionados por MSVNS permitió recuperar las clases originales a las que corresponden las proteínas (objetivo 4).

Se ha mostrado también como MSVNS es competitivo frente a otros métodos de comparación, que trabajan sobre el mismo o sobre otros modelos, tanto a nivel de resultados como a nivel de tiempos (objetivo 1). En cuanto a la utilización de MSVNS para la clasificación de proteínas, los experimentos han mostrado que, en términos del nivel de familia en SCOP y el nivel de arquitectura en CATH, los valores de superposición (normali-

zados) son suficientes para capturar la similaridad. En el caso del nivel de plegamiento en SCOP, aunque se obtuvieron mejores resultados con DaliLite, la versión utilizada de MSVNS (MSVNS2, que suponía un compromiso entre calidad de resultados y velocidad) obtuvo resultados razonablemente buenos. En general, para una instancia dada, cualquiera de los algoritmos existentes puede ser mejor que los demás. Por ello, los resultados de unos y otros algoritmos se complementan, estando MSVNS entre los mejores para todos los experimentos. Todo lo anterior supone un amplio abanico de resultados positivos en términos biológicos con los que se cumple sobradamente el objetivo 4.

Como prueba de la importancia de MSVNS hay que destacar también que ha sido aceptado e incorporado en el servidor ProCKSI [8], un servidor multi-capa de comparación de proteínas que proporciona varios métodos de comparación con objeto de comparar, analizar y mejorar las observaciones de similaridad. MSVNS fue elegido al tratarse de un método rápido y preciso para la resolución de uno de los modelos principales de comparación de proteínas: el Max-CMO.

Finalmente, siguiendo con la comparación de estructuras de proteínas, y dando cumplimiento al objetivo 3, se ha mostrado cómo los mapas de contacto difusos son un modelo adecuado para la información imprecisa asociada a los mapas de contactos y que pueden usarse para realizar la comparación de forma efectiva, tanto mediante la medida universal de similaridad (USM) como mediante una versión adaptada para el modelo GMax-FCMO del algoritmo MSVNS. Con USM hemos visto que el uso adecuado de mapas de contacto difusos puede conducir a mejores resultados de los que pueden obtenerse con los mapas estándar. Posteriormente, en la aplicación de MSVNS para la resolución del GMax-FCMO se ha comprobado que dicha aplicación es factible y que conduce a valores de similaridad apropiados. Se observó, incluso, hay instancias para las que, con la resolución del GMax-FCMO, se pueden obtener mejores valores, en términos de Max-CMO, que los obtenibles al resolver Max-CMO directamente. Esto seguramente se deba a que el espacio de búsqueda que define el modelo GMax-FCMO, con una función de costo más continua, puede llegar a ser más adecuado para los algoritmos heurísticos que el asociado a Max-CMO con su función de costo discreta. La función de costo de GMax-FCMO da lugar también a un mayor número de valores de superposición posibles, que permiten discriminar mejor entre las

distintas soluciones.

Trabajo futuro

A partir del trabajo realizado, surgen nuevas ideas, posibilidades de mejora y líneas de investigación que se plantean aquí a modo de trabajo futuro a realizar tras la terminación de la tesis.

Por un lado, el esqueleto SiGMA para generar SAD basados en optimización, es un sistema genérico, dinámico y potente. No obstante, en base a la experiencia en su uso se puede plantear la realización de algunas extensiones. Entre las posibilidades de mejora se encuentra la de permitir la ejecución de los *resolutores* en máquinas remotas (actualmente sólo se pueden ejecutar los algoritmos en la misma máquina en que está ejecutándose SiGMA). Al ser ésta una opción independiente del problema o modelo a resolver, podría incorporarse en el propio núcleo del esqueleto de SiGMA para que estuviera disponible para cualquier SAD desarrollado como extensión a SiGMA. Otra adición posible es incluir funcionalidades de análisis de resultados más avanzadas en los SAD de problemas específicos, como puede ser gestionar los clics en el árbol de resultados no sólo en las hojas sino también en niveles superiores de la jerarquía. Para los niveles superiores, el comportamiento podría ser el mostrar estadísticas de la tarea/sesión completa, tales como el mejor costo, el costo medio, el mejor algoritmo, tiempos medios para alcanzar soluciones, etc. Muchos de estos módulos para estadísticas de aspectos como los tiempos o los costos podrían realizarse de tal forma que su reutilización en problemas diferentes fuera muy sencilla, cambiando únicamente la manera en que se recuperan los valores necesarios de los ficheros de resultados generados por cada *resolutor* del problema específico, a la vez que se mantiene sin cambios la generación de gráficas y/o tablas.

En cuanto al algoritmo MSVNS hay al menos tres maneras de obtener nuevas mejoras sobre los resultados actuales: 1) mediante la búsqueda de mejores estructuras de entorno especializadas; 2) mediante un mejor ajuste de los valores de los parámetros, o su adaptación automática según las instan-

cias a resolver o la evolución de la búsqueda; y 3) comenzando la búsqueda en soluciones generadas con métodos heurísticos sencillos como GRASP, en lugar de usar sólo soluciones iniciales aleatorias. Otro plan de mejora conlleva la inclusión de una fase de preprocesado de las instancias que permita evitar las comparaciones entre estructuras que son muy diferentes, tal y como hace DaliLite. Más aún, gracias a su velocidad y simplicidad podría considerase también el uso de MSVNS como una heurística para el cálculo de las cotas inferiores en los algoritmos exactos para el modelo Max-CMO.

En lo que respecta al uso de mapas de contacto difusos quedan abiertas varias cuestiones: a) cómo realizar una elección adecuada de tipos de contacto difusos o qué funciones de pertenencia son más idóneas para conseguir mejores valores de similaridad; b) qué características concretas de las instancias pueden aconsejar su resolución mediante GMax-FCMO en lugar de Max-CMO; y c) el uso de diferentes funciones objetivo que, a la hora de evaluar ciclos que conlleven tipos de contacto diferentes, contemplen más opciones que la simple devolución de valores negativos. Una alternativa podría ser el uso de pesos positivos o negativos en función de una medida de la "compatibilidad" de los dos tipos de contacto involucrados en el ciclo. También podrían considerarse alternativas para la multiplicación de los valores de pertenencia, pudiendo estudiarse su sustitución por un mínimo, un máximo o un valor promedio entre ambos grados.

Finalmente, la observación de que ningún algoritmo heurístico es el mejor para todas las posibles instancias abre una línea de investigación especialmente interesante que es la de cómo combinar las similaridades obtenidas por métodos distintos de cara a conseguir medidas más robustas que complementen los puntos débiles de un método individual con los puntos fuertes de otros métodos, de forma que las similaridades obtenidas y la relación entre las similaridades de todas las instancias de un conjunto de datos dado sean más fiables y conduzcan a clasificaciones más acertadas. Esta línea de investigación se encuentra ya en marcha, y se espera que dé lugar a resultados interesantes durante los meses posteriores a la presentación de esta tesis.

Bibliografía

[1] R. Altman and S. Raychaudhuri. Whole-genome expression analysis: challenges beyond clustering. *Curr. Opin. Struct. Biol*, 11(3):340–347, 2001.

[2] A. Andreeva, D. Howorth, S. E. Brenner, T. J. P. Hubbard, C. Chothia, and A. G. Murzin. Scop database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Res*, 32(Database issue):D226–D229, Jan 2004.

[3] A. Andreeva, D. Howorth, J. Chandonia, S. Brenner, T. Hubbard, C. Chothia, and A. Murzin. Data growth and its impact on the SCOP database: new developments. *Nucleic Acids Research*, 36(Database issue):D419, 2008.

[4] D. Arnott and G. Pervan. A critical analysis of decision support systems research. *Journal of Information Technology*, 20(2):67–87, June 2005.

[5] Z. Aung and K.-L. Tan. Matalign: Precise protein structure comparison by matrix alignment. *Journal of Bioinformatics and Computational Biology*, 4(6):1197–1216, 2006.

[6] D. Baker and A. Sali. Protein Structure Prediction and Structural Genomics. *Science*, 294(5540):93–96, 2001.

[7] H. Barrow and R. Burstall. Subgraph isomorphism, matching relational structures and maximal cliques. *Information Processing Letters*, 4(83), 1976.

[8] D. Barthel, J. D. Hirst, J. Blazewicz, E. K. Burke, and N. Krasnogor. ProCKSI: a decision support system for Protein (Structure) Compa-

rison, Knowledge, Similarity and Information. *BMC Bioinformatics*, 8:416, 2007.

[9] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S. Eddy, S. Griffiths-Jones, K. Howe, M. Marshall, and E. Sonnhammer. The Pfam Protein Families Database. *Nucleic Acids Research*, 30(1):276–280, 2002.

[10] R. Battiti. Reactive search: Toward self–tuning heuristics. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors, *Modern Heuristic Search Methods*, pages 61–83. John Wiley & Sons Ltd., Chichester, 1996.

[11] J. D. Beltrán, J. E. Calderón, R. J. Cabrera, J. A. M. Pérez, and J. M. Moreno-Vega. Fuzzy Stopping Rules for the Strip Packing Problem. In *IPMU 2004, Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Perugia (Italia), 2004.

[12] C. Bennett, P. Gacs, M. Li, P. Vitanyi, and W. Zurek. Information distance. *IEEE Transactions on Information Theory*, 44:4:1407–1423, 1998.

[13] H. Berman, K. Henrick, H. Nakamura, and J. L. Markley. The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data. *Nucl. Acids Res.*, 35(suppl_1):D301–303, 2007.

[14] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[15] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[16] P. Bourne and H. Weissig. *Structural Bioinformatics*. Wiley-Liss, Inc, 2003.

[17] C. Bron and J. Kerbosch. Algorithm 457, finding all cliques of an undirected graph. *Communications of the ACM*, 16(575), 1973.

[18] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol*, 268(1):78–94, 1997.

[19] J. F. Campbell. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, 72(2):387–405, January 1994.

[20] A. Caprara, R. Carr, S. Istrail, G. Lancia, and B. Walenz. 1001 optimal pdb structure alignments: Integer programming methods for finding the maximum contact map overlap. *Journal of Computational Biology*, 11(1):27–52, 2004.

[21] A. Caprara and G. Lancia. Structural alignment of large-size proteins via lagrangian relaxation. In *Proceedings of RECOMB 2002*. ACM, 2002.

[22] R. Carraghan and P. Pardalos. Exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375, 1990.

[23] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, Jan. 1985.

[24] L. Chew and K.Kedem. Finding consensus shape for a protein family. In *18th ACM Symp. on Computational Geometry. Barcelona, Spain*, 2002.

[25] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, Mar. 1990.

[26] M. Dorigo. *Optimization, Learning and Natural Algorithms (en italiano)*. PhD thesis, Politecnico di Milano, Italy, 1992.

[27] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[28] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[29] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997.

[30] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[31] M. Dorigo and T. Stützle. *Ant Colony Optimization (Bradford Books)*. Bradford Books. The MIT Press, 2004.

[32] R. Durbin. *Biological Sequence Analysis: Probabalistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[33] B. Eckel. *Thinking in Java (4th Edition)*. Prentice Hall PTR, 2006.

[34] A. E. Eiben, P.-E. Raué, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 78–87, Berlin, 1994. Springer.

[35] A. E. Eiben and Z. Ruttkay. Constraint-satisfaction problems. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C5.7:1–8. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[36] I. Eidhammer, I. Jonassen, and W. R. Taylor. *Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis*. Wiley, 2003.

[37] A. T. Ernst and M. Krishnamoorthy. Efficient algorithms for the uncapacitated single allocation p-hub median problem. *Location Science*, 4(3):139–154, October 1996.

[38] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[39] D. Fischer, A. Elofsson, D. Rice, , and D. Eisenberg. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In *Pacific Symp. on Biocomputing*, pages 300 – 318, 1996.

[40] L. Fogel. Toward inductive inference automata. *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, 1962.

[41] L. Fogel, A. Owens, M. Walsh, et al. *Artificial Intelligence Through Simulated Evolution*. Wiley, 1966.

[42] J. Fruton. *Molecules and Life: Historical Essays on the Interplay of Chemistry and Biology*. Wiley-Interscience, 1972.

[43] E. Gardiner, P. Artymiuk, and P. Willett. Clique-detection algorithms for matching three-dimensional molecular structures. *J Mol Graph Model*, 15(4):245–53, 1997.

[44] D. Gilbert. JFreeChart: A free Java chart library.

[45] F. Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5):533–549, 1986.

[46] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[47] F. Glover and B. Melián. Búsqueda tabú. *Inteligencia Artificial, Revista Iberoamericana de IA*, 7(19):29–48, 2003.

[48] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[49] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms and their application*, pages 41–49, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[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] J. R. González, D. Pelta, and N. Krasnogor. Protein structure comparison through fuzzy contact maps and the universal similarity metric. In *EUSFLAT-LFA 2005*, pages 1124–1129, 2005.

[52] J. R. González, D. A. Pelta, and J. M. Moreno-Vega. Multistart VNS for the Maximum Contact Map Overlap Problem. In *Proceedings of*

the 18th Mini Euro Conference on VNS (MECVNS), Tenerife, Spain, 2005.

[53] J. R. González, D. Pelta, and M. Moreno-Vega. A simple and fast heuristic for protein structure comparison. *Enviado a BMC Bioinformatics*, 2007.

[54] J. R. González and D. A. Pelta. Una herramienta dinámica para la comparación de algoritmos. *IV Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, 2005.

[55] J. R. González and D. A. Pelta. Soft computing in bioinformatics. *21st European Conference on Operational Research (EURO XXI)*, pages 72–72, 2006.

[56] J. R. González and D. A. Pelta. On using fuzzy contact maps for protein structure comparison. *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2007.

[57] J. R. González, D. A. Pelta, and A. D. Masegosa. A Framework for developing Optimization-based Decision Support Systems. *Enviado a Expert Systems With Applications*, 2007.

[58] J. R. González, D. A. Pelta, and J. L. Verdegay. A software tool to help in the management, execution and comparison of algorithms. *Metaheuristics International Conference (MIC)*, 2005.

[59] J. Gosling and H. McGilton. The Java language environment: A white paper. Technical report, Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA, Oct. 1996.

[60] S. Goss, S. Aron, J. Deneubourg, and J. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, Dec. 1989.

[61] P.-P. Grassé. La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80, Mar. 1959.

[62] D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press New York, 1997.

[63] P. Hansen and N. Mladenovic. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, May 2001.

[64] P. Hansen, N. Mladenovic, and J. A. Moreno-Pérez. Búsqueda de entorno variable. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 19:77–92, 2003.

[65] G. Harik. Linkage learning via probabilistic modeling in the ECGA. Technical Report IlliGAL 99010, University of Illinois, 1999.

[66] W. Hart, N. Krasnogor, and J. Smith. *Recent Advances in Memetic Algorithms*. Springer, 2005.

[67] J. Holland. Adaption in natural and artificial systems, 1975.

[68] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–138, Sept. 1993.

[69] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[70] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996.

[71] Jmol: an open-source java viewer for chemical structures in 3d.

[72] K. Karlin. Metalloenzymes, structural motifs, and inorganic models. *Science*, 261(5122):701, 1993.

[73] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[74] P. Koehl. Protein structure similarities. *Current Opinion in Structural Biology*, 11:348–353, 2001.

[75] 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.

[76] N. Krasnogor. Self generating metaheuristics in bioinformatics: The proteins structure comparison case. *Genetic Programming and Evolvable Machines*, 5(2):181–201, 2004.

[77] N. Krasnogor and D. Pelta. Measuring the similarity of protein structures by means of the universal similarity metric. *Journal of Bioinformatics*, 20(7):1015–1021, May 2005.

[78] G. Lancia, R. Carr, B. Walenz, and S. Istrail. 101 optimal pdb structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. In *RECOMB '01: Proceedings of the fifth annual international conference on Computational biology*, pages 193–202, New York, NY, USA, 2001. ACM Press.

[79] G. Lancia and S. Istrail. Protein structure comparison: Algorithms and applications. In C. Guerra and S. Istrail, editors, *Protein Structure Analysis and Design,*, volume 2666 of *Lecture Notes in Bioinformatics*, pages 1–33. Springer-Verlag, 2006.

[80] E. Lander, L. Linton, B. Birren, C. Nusbaum, M. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[81] R. A. Laskowski. Structural quality assurance. In P. Bourne and H. Weissig, editors, *Structural Bioinformatics*. Wiley-Liss, Inc, 2003.

[82] K.-W. Lee and S.-Y. Huh. Model-solver integration in decision support systems: a web services approach. In *Pre-ICIS SIG-DSS Workshop*, December 2003.

[83] K.-W. Lee and S.-Y. Huh. A model-solver integration framework for autonomous and intelligent model solution. *Decision Support Systems*, 42(2):926–944, November 2006.

[84] M. Lee, G. Gippert, K. Soman, D. Case, and P. Wright. Three-dimensional solution structure of a single zinc finger DNA-binding domain. *Science*, 245(4918):635–637, 1989.

[85] N. Leibowitz, Z. Fligerman, R. Nussinov, and H. Wolfson. Multiple structural alignment and core detection by geometric hashing. In T. L.

Et.Al., editor, *Procs of 7th Intern. Conference on Intelligent Systems for Molecular Biology ISMB 99.*, pages 167–177. AAAI Press, 1999.

[86] J. Leluk, L. Konieczny, and I. Roterman. Search for structural similarity in proteins. *Bioinformatics*, 19(1):117–124, 2003.

[87] M. Li, X. Chen, X. Li, B. Ma, and P.Vitanyi. The similarity metric. In *Proc. of the 14th ACM-SIAM Symp. Discrete Algorithms(SODA) 2003*, 2003.

[88] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

[89] X. Li, D. Ruan, and A. J. van der Wal. Discussion on soft computing at flins'96. *International Journal of Intelligent Systems*, 13(2-3):287–300, 1998.

[90] H. Liisa and J. Park. DaliLite workbench for protein structure comparison. *Bioinformatics*, 16(6):566–567, 2000.

[91] P. Luan, R. Ramirez, and R. St. Louis. Expert systems that satisfy model-solver-data independence. In *System Sciences, 1990., Proceedings of the Twenty-Third Annual Hawaii International Conference on*, volume iii, pages 287–292 vol.3, 1990.

[92] S. W. Mahfoud. Crowding and preselection revisited. In R. Männer and B. Manderick, editors, *Parallel problem solving from nature 2*, pages 27–36, Amsterdam, 1992. North-Holland.

[93] B. Martín del Brío and A. Sanz Molina. *Redes Neuronales y Sistemas Borrosos. 3^a Edición*. Editorial RA-MA, 2006.

[94] S. Mason and N. Graham. Areas beneath the relative operating characteristics(ROC) and relative operating levels(ROL) curves: Statistical significance and interpretation. *Quarterly Journal of the Royal Meteorological Society*, 128(584):2145–2166, 2002.

[95] A. May. Towards more meaningful hierarchical classification of aminoacids scoring matrices. *Proteins: Structure, Function and Genetics*, 37:20–29, 1999.

[96] A. McCosh and B. Correa-Pérez. The optimization of what? In *Intelligent Decision-making Support Systems*, pages 463–481. Springer, 2006.

[97] B. Melián, J. Moreno Perez, and J. Marcos Moreno-Vega. Metaheuristicas: Una visión global. *Inteligencia Artificial, Revista Iberoamericana de IA*, 7(19):7–28, 2003.

[98] A. Miranker. Protein complexes and analysis of their assembly by mass spectrometry. *Curr. Opin. Struct. Biol*, 10:601–606, 2000.

[99] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[100] P. Mladenovic, N.and Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, Nov. 1997.

[101] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, California Institute of Technology, Pasadena, CA, 1989.

[102] P. Moscato. Memetic algorithms: a short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New ideas in optimization*, pages 219–234. McGraw-Hill Ltd., UK, 1999.

[103] H. Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In G. J. Rawlins, editor, *Foundations of Genetic Algorithms 1*, pages 316–337, San Mateo, CA, USA, 1991. Morgan-Kaufmann.

[104] H. Mühlenbein and H.-M. Voigt. Gene pool recombination in genetic algorithms. *Metaheuristics: Theory and Applications*, pages 53–62, 1996.

[105] A. Murzin, S. Brenner, T. Hubbard, and C. Chothia. Scop: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995. http://scop.mrc-lmb.cam.ac.uk/scop/.

[106] M. E. O'Kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32(3):393–404, December 1987.

[107] C. Orengo, A. Michie, S. Jones, D. Jones, M. Swindells, and J. Thornton. CATH–a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108, 1997.

[108] I. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, Oct. 1996.

[109] J. M. Pasteels, J. L. Deneubourg, and S. Goss. Self-organization mechanisms in ant societies (I): trail recruitment to newly discovered food sources. *Experientia Supplementum*, 54:155–175, 1987.

[110] L. Pauling and R. Corey. The Structure of Fibrous Proteins of the collagen-gelatin Group. *Proceedings of the National Academy of Sciences*, 37(5):272–281, 1951.

[111] F. Pearl, C. Bennett, J. Bray, A. Harrison, N. Martin, A. Shepherd, I. Sillitoe, J. Thornton, and C. Orengo. The CATH database: an extended protein family resource for structural and functional genomics. *Nucleic Acids Research*, 31(1):452–455, 2003.

[112] D. Pelta, N. Krasnogor, C. Bousono-Calzon, J. L. Verdegay, J. Hirst, and E. Burke. A fuzzy sets based generalization of contact maps for the overlap of protein structures. *Journal of Fuzzy Sets and Systems*, 152(1):103–123, 2005.

[113] L. S. Pitsoulis and M. G. C. Resende. Greedy randomized adaptive search procedures. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.

[114] D. J. Power. Specifying an expanded framework for classifying and describing decision support systems. *Communications of the Association for Information Systems*, 13:158–166, 2004.

[115] D. J. Power. A brief history of decision support systems, version 4.0. DSSResources.COM, World Wide Web, March 2007.

[116] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[117] J. Ramik. Soft Computing: Overview and Recent Developments in Fuzzy Optimization. Technical report, Institute for Research and Applications of Fuzzy Modeling, University of Ostrava, 2001.

[118] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[119] C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives: a Guide to GA Theory*. Kluwer Academic Publishers, 2003.

[120] J. Richardson, D. Richardson, N. Tweedy, K. Gernert, T. Quinn, M. Hecht, B. Erickson, Y. Yan, R. McClain, M. Donlan, et al. Looking at proteins: representations, folding, packing, and design. Biophysical Society National Lecture, 1992. *Biophysical Journal*, 63(5):1185–1209, 1992.

[121] S. Ríos-Insúa, C. Bielza Lozoya, and A. Mateos Caballero. *Fundamentos de los Sistemas de Ayuda a la Decisión*. Editorial Ra-Ma, Madrid-España, 2002.

[122] A. Sancho-Royo, J. L. Verdegay, and E. Vergara-Moreno. Some practical problems in fuzzy sets-based decision support systems. *Mathware and Soft Computing*, 6 (2-3):173–187, 1999.

[123] J. P. Shim, M. Warkentin, J. F. Courtney, D. J. Power, R. Sharda, and C. Carlsson. Past, present, and future of decision support technology. *Decis. Support Syst.*, 33(2):111–126, 2002.

[124] I. Shindyalov and P. Bourne. Protein structure alignment by incremental combinatorial extension (ce) of the optimal path. *Protein Engineering*, 11(9):739 – 747, 1998.

[125] D. Streiner and J. Cairney. What's Under the ROC? An Introduction to Receiver Operating Characteristics Curves. *Canadian Journal of Psychiatry*, 52(2):121–128, 2007.

[126] D. M. Strickland, E. Barnes, and J. S. Sokol. Optimal Protein Structure Alignment Using Maximum Cliques. *Operations Research*, 53(3):389–402, 2005.

[127] T. Stützle and H. H. Hoos. Max-min ant system. *Future Gener. Comput. Syst.*, 16(9):889–914, 2000.

[128] G. Syswerda. Simulated Crossover in Genetic Algorithms. In L. D. Whitley, editor, *Proceedings of the 2nd Workshop on Foundations of Genetic Algorithms*, pages 239–255, San Mateo, CA, USA, 1993. Morgan-Kaufmann.

[129] J. D. Szustakowsky and Z. Weng. Protein structure alignment using a genetic algorithm. *PROTEINS: Structure, Function, and Genetics*, 38:428–440, 2000.

[130] W. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Science*, 8:654–665, 1999.

[131] B. Thiruv, G. Quon, S. A. Saldanha, and B. Steipe. Nh3d: a reference dataset of non-homologous protein structures. *BMC Struct Biol*, 5, July 2005.

[132] E. Turban, J. E. Aronson, T.-P. Liang, and R. Sharda. *Decision Support and Business Intelligence Systems (8th Edition)*. Prentice Hall, 2006.

[133] J. Verdegay and E. Vergara-Moreno. Fuzzy termination criteria in knapsack problem algorithms. *MathWare and Soft Computing*, 7(2-3):89–97, 2000.

[134] J. L. Verdegay, R. R. Yager, and P. P. Bonissone. On heuristics as a fundamental constituent of soft computing. *Fuzzy Sets and Systems*, In Press:–, 2007.

[135] M. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. Bradford Books, 1999.

[136] S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors. *Metaheuristics : advances and trends in local search paradigms for optimization*. Kluwer Academic Publishers, Boston, 1999.

[137] R. A. Watson, G. Hornby, and J. B. Pollack. Modeling building-block interdependency. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, volume 1498 of

Lecture Notes In Computer Science, pages 97–108, London, UK, 1998. Springer-Verlag.

[138] P. Wright and H. Dyson. Intrinsically unstructured proteins: reassessing the protein structure-function paradigm. *J. Mol. Biol*, 293(2):321–331, 1999.

[139] J. C. Xiaobo Zhou and S. Wong. Protein structure similarity from principle component correlation analysis. *BMC Bioinformatics*, 7(40), 2006.

[140] W. Xie and N. V. Sahinidis. A branch-and-reduce algorithm for the contact map overlap problem. In *Proceedings of RECOMB 2006*, volume 3909 of *Lecture Notes in Bioinformatics*, pages 516–529. Springer, 2006.

[141] W. Xie and N. V. Sahinidis. A reduction-based exact algorithm for the contact map overlap problem. *Journal of Computational Biology*, 14(5):637–654, 2007.

[142] L. Zadeh. Fuzzy sets. *Information Control*, 8:338–353, 1965.

[143] L. A. Zadeh. Soft computing and fuzzy logic. *IEEE Softw.*, 11(6):48–56, 1994.

[144] L. A. Zadeh. Applied soft computing - foreword. *Applied Soft Computing*, 1(1):1–2, June 2001.

[145] L. A. Zadeh. Generalized theory of uncertainty (gtu)–principal concepts and ideas. *Computational Statistics & Data Analysis*, 51(1):15–46, Nov. 2006.

[146] A. Zemla. LGA: a method for finding 3D similarities in protein structures. *Nucl. Acids Res.*, 31(13):3370–3374, 2003.

[147] D. Zhi, S. S. Krishna, H. Cao, P. Pevzner, and A. Godzik. Representing and comparing protein structures as paths in three-dimensional space. *BMC Bioinformatics*, 7:460, 2006.