Universitat d'Alacant
Universidad de Alicante

Optimización de búsquedas en grandes conjuntos de datos mediante la utilización de algoritmos de *clustering* con preservación de la topología

Yuri Quintana Pacheco

Tesis Doctorales

www.eltallerdigital.com

UNIVERSIDAD de ALICANTE

Universidad de Alicante

Departamento de Tecnología Informática y Computación

Tesis doctoral

Optimización de búsquedas en grandes conjuntos de datos mediante la utilización de algoritmos de *clustering* con preservación de la topología

Autor
Lic. Yuri Quintana Pacheco

Tutor
Dr. Daniel Ruiz Fernández

Enero 2013

Al proyecto Y(uri)^{2}s

Agradecimientos

Los que me conocen saben que soy de pocas palabras, de hecho, creo que esto para mi ya es un exceso. Pero el agradecimiento es una virtud que admiro mucho, y su falta, uno de los defectos que más deploro. Así que mi agradecimiento a los que me han ayudado en este trabajo es tan sencillo como sincero.

- A mi tutor Daniel Ruiz, sin duda la persona que más me ha ayudado a llegar hasta aquí.
- A Luciano, con quien encontré el principio, y a Diego, con cuya portada llegó el final.
- A Yasset y Roberto, del CIGB, que también estuvieron en el principio y con quienes obtuve muchos materiales.
- Al Proyecto Habana y al DTIC, por el programa de doctorado dentro del que se realiza esta investigación.
- A los que siempre me halaron (o empujaron): Alberto, el Yudy, Uris y mi familia.
- Al entorno de redes, en medio de cuyo caos y con cuya fauna encontré conversación, herramientas, debates, ideas, detalles tan interesantes como inservibles ...
- A los becarios y técnicos del DTIC, siempre dispuestos a ayudarme.

Resumen

En la presente investigación se propone un método para realizar búsquedas en grandes conjuntos de datos con características similares a los grafos. El método está dividido en dos etapas: una etapa de organización del conjunto de datos y una etapa donde se puede realizar la búsqueda de elementos que minimicen una función objetivo según el modelo organizativo obtenido.

Para la organización del conjunto de datos se propone el empleo del algoritmo *growing neural gas*, al que se le han incorporado modificaciones que facilitan su empleo en el contexto de la investigación. El método de grafos embebidos en espacios vectoriales se utiliza para obtener representaciones de los elementos del conjunto de datos apropiadas para la aplicación del algoritmo de *clustering*. Como resultado del proceso de organización cada elemento del conjunto de datos es asociado a un nodo del modelo obtenido de la aplicación del algoritmo de *clustering*.

Para la etapa de búsqueda se propone un algoritmo basado en la heurística *simulated annealing*. El algoritmo realiza estimaciones de la calidad de los elementos asociados a cada nodo del modelo mediante evaluaciones de la función objetivo que realiza mientras se recorre el modelo. Estas estimaciones son empleadas para orientar la búsqueda.

El funcionamiento de cada etapa de la propuesta de solución es validada de forma empírica mediante el empleo de conjuntos de datos y funciones objetivo seleccionados con este fin. Los resultados de la aplicación del método general propuesto muestran que la obtención de soluciones requiere la evaluación de un pequeño porcentaje de los elementos del conjunto de datos, de modo que se reducen los tiempos asociados al proceso de búsqueda.

Resum

En la present investigació es proposa un mètode per a realitzar recesques en grans conjunts de dades amb característiques semblants als grafos. El mètode està dividit en dos etapes: una etapa d'organització del conjunt de dades i una etapa on es pot realitzar la recerca d'elements que minimitzen una funció objectiu segons el model organitzatiu obtingut.

Per a l'organització del conjunt de dades es proposa la utilització de l'algoritme *growing neural gas*, a què se li han incorporat modificacions que faciliten el seu us en el context de la investigació. El mètode de grafos embeguts en espais vectorials s'utilitza per a obtindre representacions dels elements del conjunt de dades apropiades per a l'aplicació de l'algoritme de *clustering*. Com resultat del procés d'organització cada element del conjunt de dades és associat a un node del model obtingut de l'aplicació de l'algoritme de *clustering*.

Per a l'etapa de recerca es proposa un algoritme basat en l'heurística *simulated annealing*. L'algoritme realitza estimacions de la qualitat dels elements associats a cada node del model per mitjà d'avaluacions de la funció objectiu que realitza mentres es recorre el model. Estes estimacions són empleades per a orientar la recerca en el model.

El funcionament de cada etapa de la proposta de solució és validada de forma empírica per mitjà de l'ocupació de conjunts de dades i funcions objectiu seleccionats amb este fi. Els resultats de l'aplicació del mètode general proposat mostren que l'obtenció de solucions requerix l'avaluació d'un xicotet percentatge dels elements del conjunt de dades, reduint-se els temps associats al procés de recerca.

Abstract

This research proposes a method to perform searches on large data sets with characteristics similar to graphs. The method is divided into two stages: a stage of organization of the data set and a stage where searches for items that minimize a function target can be performed, according to the model previously obtained.

The "growing neural gas" algorithm is proposed for organizing the data set, to which modifications have been incorporated in order to facilitate their use in the context of the investigation. The method of "graph embedded in vector spaces" is used to obtain representations of the elements of the data set suitable for the application of the clustering algorithm. As a result of the process of organization, each element in the data set is associated with a node of the model obtained from the application of the clustering algorithm.

An algorithm based on the heuristic "simulated annealing" is proposed for the searching stage. The algorithm estimates the quality of elements associated with each node of the model through evaluations of the objective function that performs while traversing the model. These estimates are used to guide the searching process.

The operation of each stage of the proposed solution is validated empirically through the use of data sets and target functions selected for this purpose. The results of the application of the proposed general method show that obtaining high quality solutions requires the evaluation of a small percentage of the data set, reducing the execution times associated with the search process.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

En los últimos 20 años se ha experimentado un crecimiento vertiginoso en la cantidad de información acumulada en diferentes sectores de la sociedad: en la sanidad, el sector empresarial, la industria farmacéutica, etc. Este crecimiento ha estado motivado, en parte, por un sentido organizativo de la sociedad, pero desde el punto de vista científico, por el reconocimiento, tanto en el propio ámbito científico como en el empresarial, del valor del conocimiento que subyace en esos datos.

En el mundo empresarial, una gran parte de los procesos han sido digitalizados, no solo para llevar un registro de las operaciones y tener un mayor control financiero o de los recursos humanos, sino para extraer información valiosa que conduzca a ganar una ventaja comercial. Así se pueden, por ejemplo, establecer relaciones entre tipos de consumidores y productos, o estimar la viabilidad de ofrecer créditos a un usuario, etc.

En el ámbito científico dos ejemplos de recopilación de información son el proyecto SETI (SETI Institute, 2012) y los proyectos genoma. El proyecto SETI ha agrupado y procesado información sobre varios tipos de señales que llegan a la tierra desde el universo, en busca de evidencia de vida extraterrestre. La información es obtenida en un observatorio donde se digitaliza, produciendo 36 Gigabytes diarios, que deben ser analizados en busca de señales anómalas, distinguibles de las emitidas por fuentes naturales.

Los distintos proyectos genoma están dirigidos a determinar la secuencia genética completa de distintos organismos, de los cuales uno

de los primeros y más importantes el proyecto Genoma Humano (BioProject, 2012). Este proyecto, conducido entre el año 1990 y el 2003, identificó los más de 20 000 genes de la especie humana, compuestos por más de 2 850 millones de nucleótidos.

Obtener estos enormes volúmenes de información, ha sido posible gracias al desarrollo tecnológico alcanzado en la informática y las telecomunicaciones, reflejado en un crecimiento exponencial en la capacidad de procesamiento de los CPU, la capacidad de almacenamiento de los discos duros y las velocidades de transmisión en las redes de datos.

En este contexto, dada la necesidad de procesar y extraer el conocimiento contenido en la información almacenada, resurgieron los campos de la estadística, el aprendizaje (*machine learning*) y el análisis de datos. Sin embargo, la combinación del volumen de información y la complejidad computacional (tanto en tiempo como en memoria) de los modelos existentes en estos campos, que en muchos casos es cuadrática o superior, hacen que sea poco factible una aplicación directa de estas técnicas. Ante esta situación emergió la necesidad de nuevas formas de solución capaces de manejar modelos complejos en intervalos de tiempo razonables, dando paso a la aparición de una nueva disciplina, la minería de datos.

Por otra parte, la diversidad de las fuentes de información ha propiciado también un desarrollo vertiginoso de las investigaciones interdisciplinarias, donde los límites entre distintas áreas científicas se han hecho difusos o han desaparecido totalmente. Nuevos campos de investigación han surgido a partir de la integración de dos o más campos, formando una nueva entidad. Ejemplos de estas nuevas áreas son la bioinformática (que combina la biología con la computación y los sistemas de información), la minería de datos (que sintetiza la estadística, la inteligencia artificial y los sistemas de bases de datos) y las modernas heurísticas (integrando ideas de diversos campos como la biología, la inmunología, la estadística y la física como fuente de inspiración para el desarrollo de técnicas de búsqueda). Estas integraciones han demostrado ser muy útiles, aportando soluciones robustas, capaces de manejar la creciente necesidad de resolver problemas reales.

Otra disciplina que debe ser considerada cuando se intenta manejar

modelos complejos y grandes volúmenes de datos es la computación paralela, pues incluso cuando se cuenta con algoritmos eficientes para el análisis de los datos, el volumen de los mismos puede implicar tiempos de procesamiento muy elevados. La computación paralela también ha tenido un desarrollo relevante en los últimos años gracias al desarrollo de la infraestructura tecnológica, tanto de *hardware* como de *software*. Entre los modelos más utilizados por los beneficios que ofrecen, especialmente en la relación prestaciones/coste, están los *clusters* de computadoras y las arquitecturas multinúcleo.

Los beneficios en la aplicación de soluciones paralelas radican en aprovechar la capacidad de cómputo disponible, distribuida en varias unidades de procesamiento o equipos. Esto se logra mediante la división de una tarea principal en subtareas, que son asignadas a las distintas unidades de procesamiento, reduciéndose de esta manera los tiempos de ejecución.

1.1 Motivación y justificación

La bioinformática y la quimioinformática se encuentran entre las áreas de una aparición relativamente reciente, a partir de la combinación de la computación y, la biología y la química, respectivamente. Se trata de áreas en desarrollo permanente por el impacto no solo científico, sino también económico y social que pueden alcanzar en esferas como la medicina y la industria química.

Uno de los retos al que se enfrentan estas disciplinas es el análisis de grandes bases de datos de compuestos químicos en el caso de la quimioinformática, y de proteínas o ácidos nucleicos en la bioinformática. Los objetivos de este análisis van desde la búsqueda de elementos específicos que den solución a una problemática concreta, hasta la exploración con el fin de detectar patrones y extraer conocimiento relevante para algún campo de aplicación.

Ambos tipos de bases de datos, de compuestos químicos y de proteínas, han experimentado un crecimiento exponencial en la última década. Dos ejemplos significativos de este crecimiento se pueden apreciar en la tabla 1.1, donde se observa el incremento en el volumen de datos que

han tenido dos bases de datos de amplio uso en la actualidad: Zinc (Zinc Database, 2012) es una base de datos de compuestos químicos comercialmente disponibles, mientras Uniprot/Swissprot (Uniprot, 2012) es una base de datos de secuencias de proteínas.

Tabla 1.1 Crecimiento de las bases de datos Zinc y Uniprot/Swissprot.

Bases de datos	**Año**	**Número de entradas**
Zinc	2005	700 000
	2007	3 300 000
	2011	13 000 000
Uniprot/Swissprot	2002	100 000
	2007	200 000
	2011	500 000

En relación a las bases de datos de compuestos químicos, una de las principales aplicaciones que requiere su análisis es la búsqueda y desarrollo de nuevos fármacos. Entre los pasos de este proceso está el *Screening Virtual*, cuyo objetivo principal es encontrar dentro de las bases de datos, micromoléculas llamadas ligandos, que interaccionen con macromoléculas, llamadas moléculas blancos o receptores (ver figura 1.1). La importancia de encontrar un compuesto que interaccione de forma adecuada con la molécula blanco radica en que este compuesto puede ser utilizado como base para una solución médica; por ejemplo, puede bloquear una región en la membrana de una célula, susceptible a ser utilizada en la penetración de algún virus.

El proceso de simulación que se realiza en el *Screening Virtual* es computacionalmente costoso (aunque esto depende del grado de precisión que se desee) por la complejidad que implica reflejar las interacciones físicas y químicas que ocurren.

También son numerosas las aplicaciones que requieren el análisis de las bases de datos bioinformáticos. Varias de estas aplicaciones incluyen la necesidad de realizar alineamiento de secuencias de proteínas, utilizado, por ejemplo, en la inferencia de la estructura espacial, en la

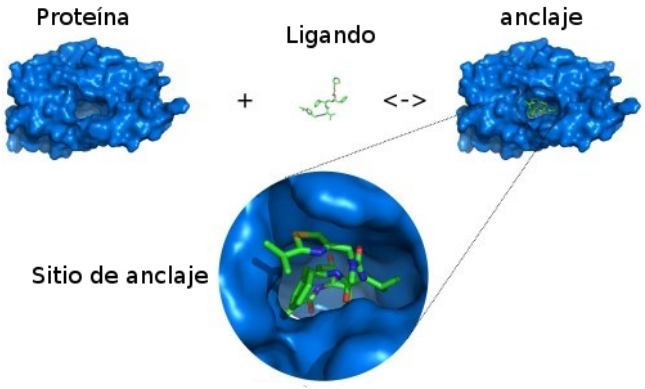


Figura 1.1 Interacción de una proteína con un ligando.

determinación de la función biológica o el establecimiento de un origen evolutivo común.

Un elemento distintivo de estas bases de datos con relación a otras más frecuentes en otros contextos como por ejemplo el empresarial, es la representación de la información. Muchas de las bases de datos empresariales presentan información numérica y textual por categorías, tipos de datos para los cuales han sido desarrolladas la mayoría de las técnicas de aprendizaje. Sin embargo, las bases de datos de compuestos almacenan moléculas, cuya representación más básica e intuitiva son los grafos. Por otro lado, las bases de datos bioinformáticos contienen cadenas de caracteres, que representan secuencias, ya sea de aminoácidos en el caso de las proteínas o bases nitrogenadas en el caso de los ácidos nucleicos como el ADN. Estas formas de representación son muy importantes pues requieren de un procesamiento distinto, ya sea por la necesidad de modificar las técnicas existentes de análisis de datos o por la necesidad de crear técnicas totalmente nuevas.

Además del reto científico que impone el análisis de las bases de datos referidas, por sus características y su volumen, la motivación para realizar este proyecto de investigación parte de una colaboración con investigadores del CIGB (Centro de Ingeniería Genética y Biotecnología). Este es uno de los principales centros de investigación de Cuba, con

resultados relevantes incluso a nivel internacional.

Uno de los principales objetivos del CIGB es el desarrollo de nuevos fármacos, enfrentándose de esta manera a una de las problemáticas descritas: el análisis de grandes bases de datos de moléculas, en la búsqueda de un candidato de solución de un problema biológico específico. Hasta el momento, en dicho centro la búsqueda es dirigida a grupos señalados de candidatos, basados en la experiencia de los investigadores, por lo que resulta una metodología bastante sesgada y dependiente de la pericia de dichos investigadores.

1.2 Formulación del problema

El ámbito del problema identificado en esta investigación lo podemos ubicar dentro del dominio general de los problemas asociados a los procesos de búsqueda de información en grandes volúmenes de datos.

Teniendo en cuenta el contexto de la investigación y los antecedentes analizados se puede definir el problema identificado en la investigación de la siguiente forma: debido al volumen considerable de información almacenado en las bases de datos, en particular en bases de datos cuyos elementos están representados en forma de grafos, y al coste computacional inherente a muchos procesos de búsqueda y evaluación que se necesitan realizar en estos, es impracticable llevar a cabo búsquedas exhaustivas.

Formalmente, y de manera general, se tiene un conjunto de datos $\mathcal{D}$, finito y discreto, y una función $f : \mathcal{D} \to \mathbb{R}$. El objetivo es encontrar $\arg\min_{x \in \mathcal{D}} f(x)$. La función $f(x)$ evalúa la calidad del elemento $x \in \mathcal{D}$ con respecto a un objetivo específico. Sin pérdida de generalidad, mientras menor es el valor de $f(x)$, mejor es la calidad del elemento x.

En el contexto de esta investigación, el conjunto $\mathcal{D}$ está formado por grafos, aunque la solución que se propone es extensible a otras formas de representación, siempre que se pueda definir una medida de distancia entre sus elementos (por ejemplo, secuencias). Los elementos de $\mathcal{D}$ están almacenados en una base de datos, $\mathcal{D}$ es un conjunto poco estructurado en el sentido que dado un elemento, hallar los cercanos a él según una métrica que se defina, implica hacer una búsqueda exhaustiva sobre

todo el conjunto, calculando la distancia a cada uno de los elementos restantes.

Cuando el conjunto está compuesto por grafos que representan moléculas, un ejemplo de f puede ser una función que evalúe la afinidad de una molécula con respecto a un blanco biológico específico. Esta función se puede calcular mediante la simulación del anclaje de la molécula al blanco biológico.

De manera similar a lo que ocurre en los problemas del campo de la optimización combinatoria no es factible hacer una búsqueda exhaustiva en $\mathcal{D}$ para hallar $\arg\min_{x\in\mathcal{D}} f(x)$. El problema es que el tiempo que tarda esta búsqueda es muy superior al tiempo que se considera razonable, o sea, que si se dispone de un tiempo T sucede que:

$$T \ll \sum_{x\in\mathcal{D}} tEval(f(x)) \tag{1.1}$$

donde $tEval$ es el tiempo que tarda la ejecución de la evaluación de la función $f(x)$ para un elemento x. El elevado valor en el tiempo de ejecución total se debe a dos causas: la elevada cardinalidad del conjunto $\mathcal{D}$ y a los relativamente altos valores de $tEval$ para las funciones que se utilizan.

Por otro lado, el hecho de que $\mathcal{D}$ sea un conjunto poco estructurado dificulta la aplicación directa de los algoritmos de búsqueda o heurísticas existentes en el campo de la inteligencia artificial (en particular aquellas basadas en búsquedas locales), pues en estos, en general, se parte de una solución factible y se explora entre las vecinas para dirigir la búsqueda.

1.3 Objetivos

El objetivo general de la investigación consiste en desarrollar un método que permita realizar búsquedas en un conjunto grande de elementos con características similares a los grafos. El propósito de las búsquedas es encontrar un elemento que minimice una función objetivo.

Para afrontar el trabajo de investigación y dar cumplimiento al objetivo global se han establecido además los siguientes objetivos parciales:

- Analizar de forma detallada la problemática relacionada con las búsquedas dentro de los conjuntos objeto del estudio.
- Identificar un tipo de organización del conjunto de datos que facilite los posteriores procesos de búsqueda, así como un mecanismo que conduzca a esta organización.
- Diseñar estrategias de búsqueda apropiadas para la estructura organizativa, utilizando mecanismos que agilicen el proceso de búsqueda y, al mismo tiempo, minimicen las posibilidades de encontrar soluciones que correspondan a mínimos locales.
- Realizar implementaciones eficientes de las técnicas utilizando estrategias de programación paralela que permitan reducir los tiempos de computación.

1.4 Hipótesis y propuesta de solución

En esta investigación se plantea como hipótesis general que para los conjuntos de grafos existen un grupo significativo de funciones que se basan en la estructura de los mismos. Esto implica que la similitud entre dos elementos se verá reflejada en algún nivel de similitud de los respectivos valores de estas funciones.

Partiendo de la hipótesis general, se asume también como hipótesis que la organización del conjunto de datos mediante la agrupación de sus elementos de acuerdo a un criterio general de similitud, permitirá establecer estrategias de búsqueda eficientes para las funciones referidas, apoyándose en la organización del conjunto de datos.

La solución que se propone en esta investigación está dividida en dos etapas. En una primera etapa se realiza una organización del conjunto de datos y en la segunda etapa se pueden realizar búsquedas independientes de elementos que minimicen una función objetivo dada.

La primera parte de la propuesta de solución consiste en la organización de la base de datos mediante el empleo de un algoritmo de *clustering* con preservación de la topología. Como resultado de la aplicación del algoritmo se obtiene un modelo que es capaz de captar la estructura topológica del conjunto de datos. Así, cada elemento del conjunto

estará asociado a un nodo del modelo, agrupando elementos similares en un mismo nodo, y a su vez, las relaciones de vecindad en el modelo reflejarán niveles de similitud entre sus elementos respectivos (ver figura 1.2).

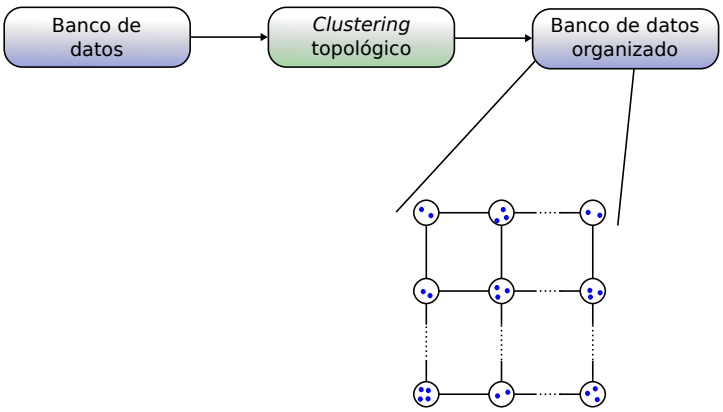


Figura 1.2 Organización de la base de datos utilizando una técnica de *clustering* que preserva la topología del espacio de entrada.

Como parte de la propuesta de solución se incluye el empleo de un método que permite una representación de los grafos adaptable para la aplicación del algoritmo de *clustering* sobre el conjunto de datos.

En la figura 1.3 se muestra el esquema general de la etapa de búsqueda. Una vez que los datos han sido organizados se pueden realizar procesos de búsqueda independientes, o sea, que empleen distintas funciones objetivos.

El proceso de búsqueda de un elemento que minimice una función objetivo se realiza mediante una heurística de búsqueda adaptada para trabajar sobre el modelo obtenido a partir de la organización de los datos. La heurística recorre el modelo haciendo evaluaciones de la función objetivo. A partir de estas evaluaciones se realiza una estimación de la calidad de los elementos de cada nodo. Estas estimaciones son empleadas por la heurística para orientar su recorrido.

La solución general que se propone tiene como objetivo ser utilizada sobre conjuntos de datos con una gran cantidad de elementos. Por esta razón en cada etapa de la propuesta se incluyen aspectos relacionados

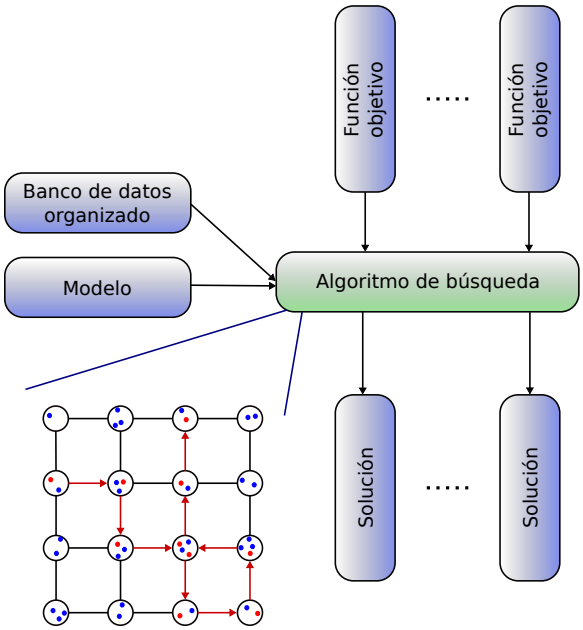


Figura 1.3 Diagrama general del procedimiento propuesto.

con la implementación en paralelo de la solución.

1.5 Estructura del documento

De aquí en adelante el documento se ha estructurado en una serie de capítulos que facilitan la comprensión del mismo.

En el capítulo 2 se realiza un estudio del estado del arte de los temas que se abarcan en la investigación. En este capítulo se incluyen aspectos relacionados con las búsquedas en grandes conjuntos de datos, los métodos de *clustering* y el paralelismo.

En el capítulo 3 se tratan aspectos vinculados al algoritmo de *clustering* que se utiliza para organizar el conjunto de datos.

El capítulo 4 se dedica a explicar el método que se utiliza para obtener representaciones vectoriales de los grafos.

En el capítulo 5 se realiza la propuesta de estrategia de búsqueda de un elemento óptimo con respecto a una función objetivo.

En los capítulos en los que se desarrolla la propuesta de solución (capítulos 3, 4 y 5) se presentan también los resultados experimentales relacionados con las respectivas propuestas.

Por último, en el capítulo 6 se exponen las principales conclusiones derivadas de la investigación, así como las líneas futuras que se avizora puedan establecerse como continuidad del trabajo.

Capítulo 2

Estado del arte

La recopilación de información es una actividad que se ha realizado a lo largo de la historia de la humanidad abarcando diversos ámbitos, desde el registro de sucesos sociales, de carácter comercial, hasta la anotación de eventos naturales.

El propósito de la recopilación varía, desde la motivación de brindar constancia para la preservación del patrimonio histórico, hasta la intención de ayudar en la comprensión de procesos tanto sociales como naturales. El proceso de recopilación se ha incrementado notoriamente con el desarrollo de la informática y las comunicaciones. Por una parte, gracias a la posibilidad técnica de digitalizar, almacenar y procesar grandes volúmenes de datos, que ha contribuido, además, a una informatización considerable de la sociedad; y por otra parte, por el desarrollo de ramas científicas encaminadas a extraer conocimiento a partir de la información.

La recopilación de información con el fin de generar conocimiento tiene un impacto significativo en la actualidad en varios sectores de la sociedad. En el área empresarial, se utiliza como una herramienta imprescindible en la toma de decisiones. Así, son estudiados comportamientos de mercados, precios, productos y usuarios, con el objetivo de encontrar patrones que permitan tomar decisiones con un beneficio comercial. Este tipo de decisiones va desde la colocación de los productos en un supermercado, la estimación de la confiabilidad de los usuarios que solicitan crédito, hasta la elección de dónde realizar inversiones en

el mercado bursátil.

En el contexto científico existen múltiples ejemplos de recopilación de información, que en muchos casos se encuentra disponible de forma pública, lo que ha constituido además un avance importante en la colaboración científica internacional.

En medicina existe un numeroso conjunto de bases de datos (Expert Health Data Programming, 2012) que son utilizadas para intentar determinar, por ejemplo, relaciones entre una enfermedad y los factores que la producen o la acentúan, ya sean genéticos, ambientales, otras enfermedades, etc.

En astronomía también se mantienen un número considerable de bases de datos con información de observaciones astronómicas (NASA, 2012; International Virtual Observatory Alliance, 2012). Uno de los proyectos de análisis de bases de datos más conocidos es el proyecto SETI (SETI Institute, 2012), que mediante el análisis de las señales de radio que llegan a la tierra busca patrones que indiquen la existencia de vida extraterrestre.

También han sido recopilados una gran cantidad de datos climatológicos (Earth System Research Laboratory, 2012), que son utilizados, entre otros propósitos, para hacer pronósticos meteorológicos y simulaciones relacionadas con el cambio climático.

En el campo de la química existen varias bases de datos de compuestos químicos (Zinc Database, 2012; BioTech FYI Center, 2012), que son analizados, entre otras aplicaciones, para el diseño de medicamentos.

En bioquímica existen bases de datos que almacenan la información obtenida sobre las secuencias que conforman las proteínas y los ácidos nucleicos (European Bioinformatics Institute, 2012). Los más reconocidos en este campo son los proyectos genoma, que tienen como objetivo determinar la secuencia genética completa de varios organismos, y entre ellos, tal vez el más importante es el proyecto Genoma Humano (Human Genome Project, 2012).

El reto que implica el manejo de estos diversos y enormes volúmenes de datos, sumado al impacto de las aplicaciones que los emplean, ha impulsado el desarrollo de las investigaciones alrededor de este tema. Existen espacios específicos dedicados al tema, ya sean publicaciones (The

VLDB Journal, 2012; Abello et al., 2002; Grossman et al., 2001) o eventos (Workshop on Algorithms for Modern Massive Data Sets, 2010). Una parte importante de las investigaciones se realizan dentro del ámbito de la inteligencia artificial, fundamentalmente en el desarrollo de técnicas de aprendizaje, enfocándose en la eficiencia de los algoritmos (Chawla, 2008). También el desarrollo ha venido por la vía de la computación de alto rendimiento, en particular, a partir del desarrollo de algoritmos paralelos (Palma et al., 2008). Además, debido a la diversidad de las fuentes de datos, han surgido algunos círculos científicos especializados en el procesamiento de bases de datos específicas, ya sean bioinformáticas, químicas, astronómicas, etc.

2.1 Recuperación de objetos en bases de datos de grandes dimensiones

Desde el punto de vista de los sistemas de información, una de las acciones a realizar sobre una base de datos es la búsqueda de uno o varios elementos específicos a partir de una solicitud (*query*). Esta tarea se hace especialmente costosa dada las enormes dimensiones de algunas bases de datos y la complejidad de los datos que almacenan.

Una de las alternativas orientadas a realizar procesos de búsqueda de elementos de manera eficiente que ha alcanzado un desarrollo relevante son los sistemas de gestión de bases de datos. Estos sistemas permiten la creación, mantenimiento y uso de bases de datos, y pueden utilizar una variedad de modelos, entre los cuales el modelo relacional es el más utilizado. Sin embargo, los sistemas de bases de datos tienen utilidad en un contexto limitado, pues aún cuando, en general, la información se almacena utilizando estructuras avanzadas (árboles balanceados, B-trees, B++ *trees*) (Powell, 2005), la información que se puede extraer de esta es más bien directa, buscando rangos específicos de valores cuantitativos o cadenas específicas. Además, en algunos casos, para obtener resultados es imprescindible realizar una búsqueda exhaustiva en los datos.

Otra alternativa utilizada cuando se necesitan realizar búsquedas en bases de datos de grandes dimensiones es utilizar formas de representación de la información que pueda ser procesada de manera eficiente, en

particular cadenas binarias. Ejemplos de este tipo de técnica se pueden apreciar en el análisis de las bases de datos de moléculas, donde estas inicialmente son almacenadas mediante algún formato descriptivo de su composición (SMILES, SDF) (Weininger, 1990; Dalby et al., 1992), a partir del cual se puede reproducir su estructura en forma de grafos; y sobre estos se aplican operaciones para obtener representaciones binarias como las llaves estructurales y los *fingerprints* (Flower, 1998; Xue et al., 2003; Munk Jørgensen et al., 2005). En el caso de las primeras, cada bit de la cadena representa la presencia o ausencia de un rasgo estructural específico (o patrón). Para la conformación de los *fingerprints* se examina la molécula y se generan patrones para rasgos estructurales (usualmente caminos sobre el grafo de la molécula). Estos patrones son utilizados como semillas para la generación de un número aleatorio que es convertido en binario, y proyectado sobre una cadena binaria. Estos *fingerprints* tienen como característica básica que si un patrón está presente en dos moléculas, en ambos *fingerprints* los bits correspondientes a ese patrón estarán activados.

También con el fin de optimizar las búsquedas de elementos en grandes conjuntos de datos se han empleado variantes que se apoyan en técnicas provenientes de la inteligencia artificial, en particular las relacionadas con el descubrimiento de conocimiento en los datos (*Knowledge Discovery in Data*). Un ejemplo que ha sido ampliamente utilizado en sistemas de recuperación de documentos son los métodos de recuperación basada en *clusters* (*cluster-based retrieval*) (Salton, 1991; Liu and Croft, 2006; Altingovde et al., 2008; Manning et al., 2008). En estos métodos los documentos son agrupados de acuerdo a su similitud. Cuando se realizan búsquedas relacionadas con una solicitud particular esta es comparada con elementos representativos de la base de datos en lugar de con todos los elementos. Por supuesto, la efectividad del método va a depender de cuán bueno sea el *clustering* que se obtenga.

2.2 *Clustering*

Una de las disciplinas que ha estado más vinculada a la evolución de las grandes bases de datos es la minería de datos (*data mining*). La minería

de datos es un campo de surgimiento relativamente reciente dentro de las ciencias de la computación. Se considera minería de datos el proceso de descubrimiento de patrones desconocidos en un conjunto grande de datos (Sumathi and Sivanandam, 2011) y utiliza, fundamentalmente, métodos provenientes de la estadística y la inteligencia artificial.

Uno de los problemas centrales en la minería de datos es resumir cantidades enormes de información en categorías más simples, más compactas y más comprensibles. La forma común y mejor estudiada de lograrlo es dividir los datos en grupos llamados *clusters*, de manera que los miembros del mismo *cluster* sean tan similares como se pueda, mientras que los miembros de distintos *clusters* sean lo más diferentes posibles. Este proceso de agrupar los datos se conoce como *clustering*. Mediante el examen de las propiedades de elementos de un mismo *cluster* se pueden encontrar reglas y conceptos que permiten caracterizar y categorizar los datos.

Dentro de la terminología asociada a los procesos de *clustering* existen varios aspectos que reflejan características de los distintos métodos, que deben ser tomados en cuenta para determinar cuál emplear en determinado problema. Una de las clasificaciones más empleadas separa los métodos de *clustering* en jerárquicos y particionales. Cuando se utilizan métodos de *clustering* jerárquico se obtiene una jerarquía donde cada nivel representa una partición del dominio, mientras que los métodos particionales dan como resultado una sola partición de los datos.

Los métodos de *clustering* jerárquico más empleados con propósitos generales son los métodos jerárquicos aglomerativos sin solapamiento, SAHN (*Sequential Aglgomerative Hierarchical Non-overlapping*) (Downs and Barnard, 2002). Los SAHN incluyen diversas variantes entre las que se encuentran *single link*, *complete link*, *average link* y Ward (Larose, 2005). Además, existen los métodos jerárquicos divisivos, entre los cuales el método de partición recursiva (Chen et al., 1998) y el método de mínimo diámetro (Guha et al., 1998) se han utilizado con bastante efectividad (Downs and Barnard, 2002).

Los métodos aglomerativos presentan 3 desventajas fundamentales (Abbass et al., 2002). En primer lugar, los *clusters* no son generados de manera natural y, por tanto, debe ser utilizado otro criterio para dete-

ner el proceso de agrupamiento e interpretar los resultados. En segundo lugar, para conjuntos de datos grandes, la forma de los *clusters* que son generados por estos métodos puede ser bastante irregular, de manera que intentar deducir caracterizaciones de los elementos miembros del *cluster* puede ser en extremo difícil. En tercer lugar, y posiblemente la desventaja más grave para las aplicaciones de minería de datos, los métodos jerárquicos, en general, requieren un tiempo cuadrático. Esta situación está esencialmente relacionada con que los algoritmos aglomerativos tienen que extraer la distancia más pequeña de un conjunto dinámico que originalmente tiene un tamaño cuadrático.

Los métodos de *clustering* particionales (*partitioning methods*) presentan una gran diversidad y entre los más empleados se encuentran: k-vecinos más cercanos, los métodos de reubicación (*k-means*, *k-medoids*) y los métodos basados en densidad (DBSCAN, OPTICS) (Han and Kamber, 2006; Xu and Wunsh, 2010).

Los métodos particionales, en general, usan una función de evaluación como guía para un mecanismo de búsqueda que genera buenos candidatos de *clusters*. Los mecanismos de búsqueda de la mayoría de los métodos particionales emplean variantes de la estrategia *hill-climbing*. La diferencia fundamental entre los algoritmos radica en la selección del criterio de optimización (Abbass et al., 2002).

Los criterios de optimización de todos los métodos particionales hacen asunciones, explícita o implícitamente, en relación con la distribución de los datos. Sin embargo, algunos métodos son más generalizables en su aplicación que otros por las asunciones que toman, mientras que otros pueden ser guiados por criterios de optimización que permiten una evaluación más eficiente.

Una de las ventajas de los métodos particionales de *clustering*, con respecto a los jerárquicos, radica en que el criterio de optimización es apropiado para una posterior interpretación de los resultados. Sin embargo, la familia de los métodos particionales incluye miembros que presentan una complejidad temporal desde lineal hasta superior a cuadrática. La razón principal para esta variabilidad reside en la complejidad del criterio de optimización que se emplea. Mientras más complejo es el criterio de optimización, más robusto al ruido será el método, pero también

más costoso computacionalmente.

2.2.1 *Clustering* sobre grandes volúmenes de datos

Las técnicas de *clustering* han surgido en distintos contextos y, a pesar de ello, existe entre estos métodos cierto nivel de similitud. Sin embargo, no todos los métodos son apropiados en todos los contextos. En particular los métodos de *clustering* apropiados para las fases exploratorias e iniciales de la minería de datos deben cumplir con las siguientes cualidades (Estivil-Castro, 2002)

- Genericidad. Casi todos los métodos de *clustering* deben estar caracterizados por dos componentes: un mecanismo de búsqueda que genera los *clusters* candidatos y una función de evaluación de la calidad de los candidatos. De esta manera, la función de evaluación hace uso de una función que permite medir la similitud entre un par de puntos de los datos. Los métodos de *clustering* pueden ser considerados genéricos si pueden ser aplicados sobre una variedad de dominios mediante la simple sustitución de la medida de similitud.

- Escalabilidad. Para ser capaces de manejar grandes volúmenes de datos, los métodos de *clustering* deben ser tan eficientes como sea posible en términos de tiempos de ejecución y necesidad de almacenamiento en memoria. Dado un conjunto de datos que incluye n elementos, cada uno con a atributos, la complejidad espacial y temporal debe de ser subcuadrática en n, y tan baja como sea posible en a (si es posible, lineal). En particular, el número de evaluaciones de la función de similitud debe mantenerse en un nivel tan bajo como sea posible. Muchas de las técnicas de *clustering* clásicas son inapropiadas para aplicaciones de minería de datos que involucren un volumen elevado de información por estar caracterizadas por una complejidad temporal cuadrática.

- Incremental. Aún cuando un método de *clustering* sea escalable, los tiempos de ejecución serán largos cuando el volumen de datos sea grande. Por esta razón es importante usar métodos en los cuales

el acercamiento a las soluciones sea de manera incremental. Los métodos incrementales permiten a los usuarios vigilar el progreso de las soluciones, y terminar la ejecución cuando se encuentra una solución con la calidad necesaria.

- Robustez. Un método de *clustering* debe comportarse de manera robusta con respecto al ruido y a los *outliers*. Ningún método es inmune a los efectos de datos erróneos, pero es una característica deseable que la presencia de ruido no afecte de manera importante los resultados.

Encontrar métodos de *clustering* que cumplan con todas estas características se mantiene hoy en día como uno de los retos de la minería de datos. En particular, muy pocos métodos son escalables a bases de datos con muchos elementos, y aquellos que son escalables no son robustos.

Entre las líneas generales que se siguen para lograr que los métodos de *clustering* sean aplicables sobre grandes conjuntos de datos están la selección de muestras aleatorias (*random sampling*) y la concentración de los datos (*data condensation*). A continuación se reseñan algunos métodos representativos de estos enfoques.

CLARA (*Clustering Large Applications*) es un método desarrollado a partir de PAM (*Partition Around Medoids*) que es una variante del algoritmo *k-medoids* (Pakhira, 2008). Este método se basa en tomar solo una pequeña porción de los datos como representación de estos, en lugar de la totalidad del conjunto, y aplicar PAM a este subconjunto. Si la muestra es seleccionada de manera suficientemente aleatoria, todo el conjunto de datos debe estar correctamente representado, y los objetos representativos (o sea, los *medoids*) elegidos deben ser similares a los que se seleccionarían para todo el conjunto de datos. El método elige varias muestras del conjunto de datos, aplica PAM a cada una y el resultado que ofrece es el mejor *clustering* obtenido.

La efectividad de CLARA depende del tamaño de las muestras. Resulta claro que si bien PAM busca por los mejores *k-medoids* para un conjunto de datos dado, CLARA busca por los mejores *k-medoids* entre las muestras seleccionadas del conjunto de datos. CLARA no puede encontrar el mejor *clustering* si alguno de los mejores *k-medoids* no fue

seleccionado en una muestra. Esta es la compensación a pagar por la eficiencia.

Con el objetivo de mejorar la calidad y escalabilidad de CLARA, otro algoritmo de *clustering* llamado CLARANS (*Clustering Large Applications based upon RANdomized Search*) fue propuesto en (Pakhira, 2008). Este es también un algoritmo basado en *k-medoids* y combina la técnica de muestreo con PAM. Sin embargo, a diferencia de CLARA, CLARANS no se restringe a alguna muestra en un momento dado. Mientras CLARA tiene una muestra fija en cada etapa de la búsqueda, CLARANS selecciona una muestra con cierta aleatoriedad en cada etapa. El proceso de *clustering* puede ser presentado como una búsqueda en un grafo donde cada nodo es una solución potencial, o sea, un conjunto de *k medoids*. El *clustering* obtenido luego de remplazar un *medoid* es llamado vecindad del *clustering* actual. El número de vecinos que serán aleatoriamente probados está limitado por un parámetro. Si un vecino mejor en términos de calidad es encontrado, CLARANS se mueve hacia él y recomienza el proceso; en caso contrario el *clustering* actual es un óptimo local. Si se encuentra un óptimo local CLARANS comienza con un nuevo conjunto de nodos seleccionados aleatoriamente. Se ha demostrado experimentalmente que CLARANS es más efectivo que PAM y que CLARA (Han and Kamber, 2006).

El método BIRCH (*Balanced Iterative Reducing and Clustering using Hierarchies*) (Zhou and Bin, 2010) introduce los conceptos de *clustering feature* y *CF tree* (*Clustering Feature tree*). Este último es un árbol que se utiliza como forma de representación resumida de los *clusters*, con el objetivo de lograr una mayor velocidad de procesamiento y escalabilidad en el proceso de aplicación de *clustering* a grandes bases de datos. El *clustering feature* es esencialmente un resumen estadístico del *cluster*, que registra medidas de interés para el cómputo de los *clusters* y utiliza el espacio computacional de forma eficiente, puesto que no es necesario almacenar la información de todos los puntos. El *CF tree* es un árbol de altura balanceada que almacena los *clustering features*. Está sujeto a dos parámetros, el factor de ramificación y el umbral de diámetro, que regulan el crecimiento del árbol.

En (Estivil-Castro, 2002) se investiga sobre un balance entre escalabilidad y robustez, mediante métodos de *clustering* que buscan imitar el comportamiento de los métodos robustos tanto como se pueda, mientras establecen un límite al número de evaluaciones de similitud entre los elementos de los datos. Las funciones de similitud entre puntos de los datos generalmente satisfacen las condiciones de una métrica. Por esta razón, se presenta un análisis basado en la conveniencia de considerar las evaluaciones de estas funciones en términos de cálculo de vecinos más cercanos en un espacio métrico apropiado. De esta manera se analiza el problema de encontrar eficientemente un *clustering* robusto reduciéndolo al problema de recopilar eficientemente la información de proximidad. Finalmente se proponen dos heurísticas basadas en *hill-climbing*, que recopilan información de vecinos más cercanos mientras se mantiene un número razonable de cálculos de distancia.

2.2.2 Algoritmos de *clustering* con preservación de la topología

Existe un grupo de algoritmos de *clustering* que tienen la característica de preservar el orden topológico de los elementos del espacio de entrada. Esto significa representar los puntos del espacio de entrada, generalmente multidimensional, como puntos en un espacio de dimensión pequeña, usualmente dos o tres, de manera tal que las distancias entre los puntos en este espacio se correspondan con las disimilitudes entre los puntos en el espacio original. Estos métodos han sido empleados en aplicaciones con el objetivo de visualizar la estructura de datos multidimensionales, y también como forma de reducción de dimensiones en conjuntos de datos (Duda et al., 2001).

Uno de estos métodos es el *multidimensional scaling* (MDS) (Duda et al., 2001), que intenta ubicar proyecciones de los elementos del espacio de entrada en un espacio de menor dimensión, tratando de conservar las distancias en el espacio original. Aunque existen varios enfoques de este problema, generalmente se formula como un problema de optimización donde la función objetivo es:

$$\min_{x_1,...,x_n} \sum_{i<j} (\|x_i - x_j\| - \delta_{i,j})^2 \tag{2.1}$$

donde $x_1, ..., x_n$ son las proyecciones de los elementos del espacio de entrada y $\delta_{i,j}$ son las distancias entre los elementos i y j.

Un método relacionado con MDS son los mapas autoorganizativos (*Self- Organizing Maps*) o mapas de Kohonen (Kohonen, 2001). Este método es uno de los modelos de redes neuronales artificiales más utilizados y es capaz de proyectar espacios multidimensionales en un espacio de dos o tres dimensiones. Desde el punto de vista de una técnica de *clustering*, los *clusters* pueden ser identificados por las coordenadas topológicas de las neuronas. Este método es particularmente útil cuando existe un mapeado no lineal inherente al problema en cuestión.

Los mapas autoorganizativos tienen una fase de entrenamiento durante la cual se construye el mapa usando los elementos de entrada, mediante un proceso competitivo (*vector quantization*). Una vez obtenido el mapa se pueden clasificar nuevos vectores de entrada en alguno de los nodos del mismo.

Con el objetivo de mejorar su utilidad en aplicaciones de minería de datos se han propuesto diversas modificaciones para los mapas autoorganizativos. Uno de los propósitos más comunes es eliminar la necesidad de definir de manera previa el tamaño de los mapas, como se aprecia en los métodos *Incremental Grid Growing* (Qiang et al., 2010a), *Growing Grid* (Ayadi et al., 2007), *Growing SOM* (Alahakoon et al., 2000), *Hypercubical* SOM (Bahuer and Villmann, 1997), *Growing Cell Structures* (Pai et al., 2007) y *Growing Hierarchical Self-Organizing Map* (Rauber et al., 2002).

El método *Incremental Growing Grid* permite agregar nuevas neuronas en los bordes del mapa; además, las conexiones entre las neuronas pueden establecerse y eliminarse de acuerdo a umbrales establecidos basados en la similitud de los vectores modelos respectivos. Esto puede conducir a mapas con una estructura irregular y no conexa. Con un espíritu similar, los *Growing* SOM utilizan un factor de expansión para controlar el proceso de crecimiento del mapa.

El método *Growing Grid* añade filas y/o columnas durante el proceso

de entrenamiento, empezando por un mapa de 2×2 neuronas. La decisión de dónde efectuar la inserción se basa en una medida que se calcula para cada neurona. Como extensión de este método, el *Hypercubical* SOM permite un proceso de crecimiento del mapa en más de dos dimensiones.

En general, se puede afirmar que estos métodos también están enfocados en lograr una distribución equitativa de los patrones de entrada en el mapa mediante la inserción de nuevas neuronas en la vecindad de las neuronas que representan un número desproporcionado de datos de entrada. Así, no reflejan de manera primaria el concepto de representación considerando una cuantificación del error, sino por la cantidad de entradas en áreas específicas del mapa.

Los *Growing Cell Structures* (GCS) son otro modelo basado en los mapas autoorganizativos, aunque los parámetros de adaptación son constantes (no varían en el tiempo) y solo la neurona ganadora y sus vecinos topológicos directos son actualizados. Además incluyen un proceso de inserción de unidades (neuronas) basado en determinar las unidades que más se han activado e insertarlas entre esta y aquella de sus vecinas que se encuentre más lejana.

Los *Growing Hierarchical Self-Organizing Maps* (GHSOM) presentan una estructura jerárquica de varias capas, donde cada capa consiste en varios SOM independientes. De manera similar al modelo de *Growing Grid*, se comienza en un mapa de primer nivel, que crece en tamaño para representar los datos con un nivel específico de detalle. Tras haber alcanzado un nivel de granularidad en la representación de los datos, las neuronas son analizadas para comprobar si representan los datos con un nivel de granularidad mínimo. Aquellas neuronas que representan datos muy diversos son expandidas para formar un nuevo SOM creciente en la capa siguiente, donde los datos deben ser representados con mayor detalle.

Las redes GNG (*Growing Neural Gas*) (Gancev and Kulakov, 2009) son otro tipo de método capaz de aprender las relaciones topológicas en un grupo de entradas mediante reglas de aprendizaje de tipo Hebbiano. Este algoritmo está relacionado con el NG (*Neural Gas*) (Martinetz and Shulten, 1991) que utiliza un esquema de aprendizaje Hebbiano competitivo (CHL, *Competitive Hebbian Learning*) (Martinetz, 1993). La idea

principal del método es añadir nuevas unidades a una red inicialmente pequeña, mediante la evaluación de medidas estadísticas locales recolectadas durante las etapas previas de adaptación. Una ventaja sobre el método *Neural Gas* es el carácter incremental del modelo, lo que elimina la necesidad de especificar un tamaño para la red como parámetro. De esta manera, el proceso de crecimiento de la red puede continuar hasta alcanzar algún criterio definible por el usuario. Todos los parámetros del método son constantes en el tiempo en contraste con otros modelos que se sustentan en parámetros decrecientes (como en el caso de los mapas autoorganizativos y las redes *Neural Gas*). También han sido propuestas extensiones de este método capaces de superar las limitaciones del mismo en cuanto a la eliminación del ruido (Hebboul et al., 2011).

2.2.3 *Clustering* sobre grafos

En ocasiones la información es representada en forma de grafos, o sea, como una colección de nodos y aristas entre nodos. Esta es una de las formas más generales de representación de datos, puesto que con ella se pueden representar entidades, sus atributos y sus relaciones con otras entidades. Algunos dominios donde esta forma de representación es utilizada son:

- Datos químicos. Cuando se modelan datos químicos los nodos se corresponden con los átomos, mientras que las aristas se corresponden con los enlaces físico-químicos que se establecen entre los átomos. En algunos casos, también algunas subestructuras muy comunes o relevantes pueden ser utilizadas como nodos. En general, los grafos individuales son bastante pequeños.

- Datos biológicos. Los datos biológicos son modelados de una manera similar a los datos químicos. Sin embargo, cada grafo individual es típicamente más grande. Por ejemplo, en el caso de las representaciones de una molécula de ADN, puede tener cientos de nodos, que serían los nucleótidos.

Muchos de los algoritmos convencionales de *clustering* pueden ser extendidos al dominio de los grafos. Las principales modificaciones que

se requieren para realizar esta extensión son las siguientes (Aggarwal and Wang, 2010):

- La mayoría de los algoritmos de *clustering* clásicos generalmente utilizan una función de distancia como medida de similitud entre elementos. Por tanto, se necesitan medidas apropiadas que definan el grado de similitud (o distancia) entre objetos estructurados.

- Muchos de los algoritmos clásicos de *clustering* emplean elementos u objetos representativos como centroides como parte de su funcionamiento. Para el caso de los objetos multidimensionales estos elementos son bastante fáciles de diseñar, sin embargo, es una tarea compleja cuando se trabaja con grafos. Por tanto, se requiere diseñar métodos apropiados que generen objetos representativos de un conjunto de grafos.

El proceso de evaluar la similitud estructural entre dos grafos también se conoce como *matching* de grafos (*graph matching*). Se han propuesto una gran variedad de métodos para tratar problemas específicos del *matching* estructural (Conte et al., 2004). Los métodos de *matching* de grafos pueden ser clasificados en sistemas de *matching* exactos y sistemas con tolerancia a fallos (Bunke and Neuhaus, 2007). A pesar de que los sistemas de *matching* exacto de grafos ofrecen un modo riguroso para describir el problema del *matching* de grafos en términos matemáticos, en general, es solo aplicable a un número restringido de problemas del mundo real. Por otra parte, los sistemas de *matching* de grafos con tolerancia a fallos son capaces de lidiar con las distorsiones intraclases, muy frecuentes en este tipo de problemas, aunque son computacionalmente más ineficientes.

La distancia de edición es una de las medidas de similitud sobre grafos más intuitivas. Sin embargo, la aplicabilidad del *matching* de grafos con tolerancia a fallos usando distancia de edición tiene una fuerte dependencia de la definición de costos de edición apropiados.

2.3 Minería de datos en paralelo

Debido a las dimensiones de las bases de datos o la complejidad de las técnicas, la minería de datos de alto rendimiento (*high performance data mining*) es una necesidad crítica. Esto implica aprovechar las ventajas que ofrecen los avances alcanzados en las arquitecturas de procesamiento. La incorporación de elementos de procesamiento permite analizar más datos, construir más modelos y mejorar los resultados de los modelos.

2.3.1 Arquitecturas paralelas

Existen diversas clasificaciones de las arquitecturas paralelas. Una de ellas está basada en el tipo de memoria usada y el grado de conectividad entre las unidades de cómputo (ver figura 2.1). Por una parte se encuentran las tecnologías multinúcleo y multiprocesadores, donde se tienen múltiples unidades de procesamiento en una misma computadora, mientras que en los *clusters*, MPPs (*massively paralell processors*) y GRIDS se emplean múltiples computadoras conectadas a través de una red.

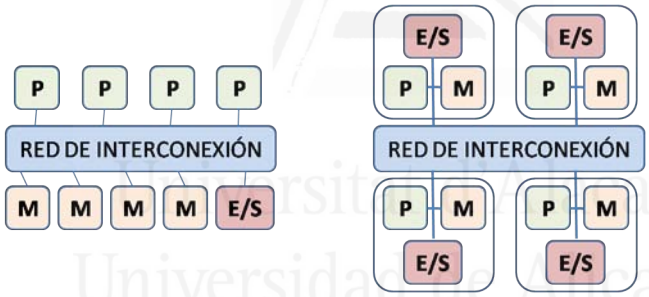
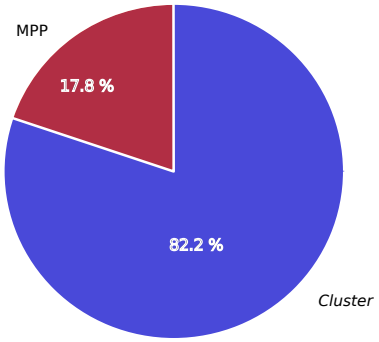


Figura 2.1 Tipos de paralelismo: multiprocesadores (izquierda) y multicomputadores (derecha).

Los multiprocesadores con memoria centralizada (*Symetric Mutli-Processors*, SMP) tienen como ventaja que requieren una programación más sencilla pues no es necesario distribuir código y datos; mientras que su principal desventaja es la escalabilidad. Contrariamente, cuando se utiliza una red de computadoras la principal ventaja con que se cuenta

es la escalabilidad, mientras que la programación se complica porque se deben manejar aspectos como la necesidad de comunicación mediante el traspaso de mensajes, la sincronización y la distribución de la carga de trabajo. Además, el rendimiento puede verse afectado por la demora en las comunicaciones a través de la red.



Fuente: top500, Noviembre 2012. http://www.top500.org

Figura 2.2 Distribución de las arquitecturas utilizadas en el procesamiento de alto rendimiento.

La alternativa que mayor popularidad ha alcanzado son las arquitecturas de tipo *cluster* (ver figura 2.2), lo cual está influenciado fundamentalmente por su escalabilidad y su capacidad de actualización. Este último aspecto es de gran importancia pues una computadora que se considera potente hoy, en unos pocos años será obsoleta. Debido a esto, la posibilidad de añadir nuevos elementos de procesamiento se convierte en una necesidad, así como eliminar los más obsoletos. Esto es posible de llevar a cabo a cierto nivel con las computadoras SMP teniendo en cuenta las limitaciones de escalabilidad y que solo se pueden añadir procesadores compatibles. Aunque en teoría las computadoras MPP no imponen restricciones de escalabilidad, las configuraciones son difíciles de cambiar, impidiendo mantener la tecnología actualizada; y a esto se suma el alto coste de este tipo de computadoras. Mientras, las computadoras SMP se hacen populares por sus precios y potencia, y junto a la facilidad de añadir este tipo de computadoras a una red, convierten a los *clusters* de computadoras SMP en una opción muy difundida.

2.3.2 Tipos de paralelismo

Existen dos tipos fundamentales de paralelismo, el paralelismo de datos y el paralelismo funcional. El paralelismo de datos se enfoca en la distribución de los datos entre los distintos nodos para su procesamiento. El paralelismo funcional, en cambio, se basa en la distribución de tareas (funciones, instrucciones, etc.) entre los nodos para su procesamiento en paralelo, y se puede ubicar en varios niveles (programa, función, bloques u operaciones).

Paralelismo intermodelo e intramodelo

En general, los procesos de minería de datos comienzan por tomar una muestra de datos y construir un modelo. La elaboración de modelos complejos con muchas variables, aún cuando la muestra de datos no sea muy grande, puede ser computacionalmente costosa y, por tanto, resulta claro apreciar los beneficios que puede conllevar el paralelismo. Además, en ocasiones se requieren análisis adicionales o la construcción de diferentes modelos para seleccionar el mejor.

En este contexto existen dos tipos de paralelismo: paralelismo intermodelo y paralelismo intramodelo (ver figura 2.3) (Small and Edelstein, 1997). El paralelismo intermodelo es un método donde múltiples modelos se crean de manera concurrente y cada uno es asignado a una unidad de procesamiento distinta. La idea es que mientras se tengan más procesadores, más modelos pueden ser construidos sin que disminuya el rendimiento. En este sentido, se aprovecha la escalabilidad con respecto a la cantidad de modelos independientes que pueden ser construidos.

El paralelismo intramodelo es un método donde un modelo es construido utilizando varios procesadores. Esta variante es especialmente apropiada cuando la construcción del modelo toma un tiempo considerable. El modelo es dividido en tareas que son ejecutadas en distintos procesadores, y los resultados son combinados para el resultado final.

2.3.3 *Clustering* en paralelo

Los beneficios que ofrece el paralelismo al aplicarse a técnicas de minería de datos han sido demostrados a través de muchos trabajos (Taniar and

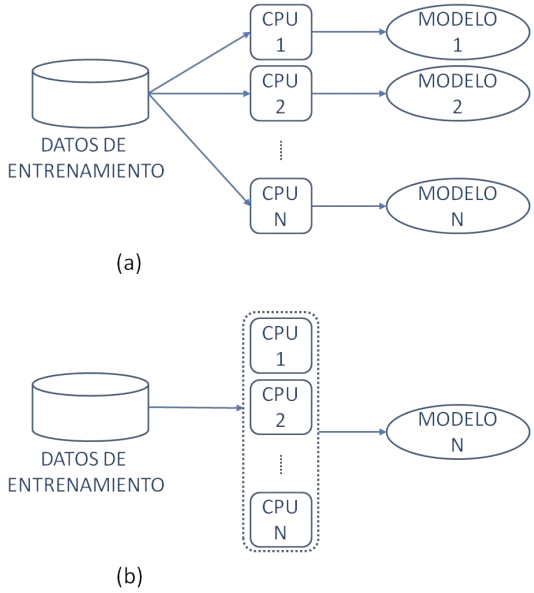


Figura 2.3 Tipos de paralelismo. (a) intermodelo, (b) intramodelo.

Rahayu, 2002; Einakian and Ghanbari, 2006; Khan, 2008), incluyendo, por supuesto, su aplicación a las técnicas de *clustering* (Foti et al., 2000; Wang et al., 2004; Hadjidoukas and Amsaleg, 2008).

Uno de los métodos de *clustering* más populares y que mejores resultados ha aportado es *K-means*, y es por esto que también se han desarrollado versiones en paralelo de este algoritmo (Zhang et al., 2006). Un ejemplo ilustrativo por su sencillez es el propuesto en (Dhillon and Modha, 1999) y cuyo pseudocódigo se muestra en el algoritmo 2.1.

En el cálculo de distancia (paso 2.), como cada proceso se concentra en la porción de los datos asignada al mismo, el procedimiento es inherentemente paralelo, o sea, que se puede ejecutar de forma asíncrona y en paralelo para cada elemento de los datos. En este contexto, una simple pero efectiva estrategia de paralelismo es dividir los datos de manera que cada proceso reciba una cantidad similar de elementos, para tratar que la carga computacional esté igualmente distribuida entre los procesadores. Sin embargo, el aspecto que puede afectar el tiempo total está asociado al costo de comunicación impuesto en el paso de recálculo de los centroides. Antes de cada nueva iteración de *k-means*, que comien-

Algoritmo 2.1 *K-means* en paralelo.

1. *Proceso de inicialización.*

   Identificar el número de procesos.

   Para cada proceso:

   Seleccionar k centroides iniciales.

   Transimitir el centroide al resto de los procesos.

2. *Cálculo de distancia: cada proceso se concentra en la porción de datos asignada al mismo.*

   Para cada elemento de los datos:

   Calcular la distancia euclidiana a cada centroide.

   Encontrar el centroide más cercano.

3. *Recálculo de los centroides.*

   Recalcular los centroides de los *clusters* como el promedio de los elementos de los datos asignados al proceso.

4. *Condición de convergencia*

   Repetir los pasos 2 y 3 hasta que se alcance la condición de convergencia.

za en el cálculo de las distancias, todos los procesos deben comunicarse para recalcular los centroides.

Este procedimiento de sincronización se realiza en el paso de recálculo de los centroides (paso 3.). En el paso de la condición de convergencia (paso 4.) se garantiza que cada proceso mantiene una copia local del total del error cuadrático medio y, por tanto, puede decidir de manera independiente el cumplimiento de la condición de convergencia. En conclusión, cada iteración de este algoritmo de *k-means* en paralelo está conformada por una fase de cómputo asíncrona, seguida por una fase de comunicación síncrona.

Otro ejemplo de algoritmo de *clustering* del que se han propuesto variantes paralelas son los mapas autoorganizativos (Arroyave et al., 2002). Estos son particularmente apropiados con esta intención gracias a que el procesamiento que se realiza en las neuronas del mapa es independiente.

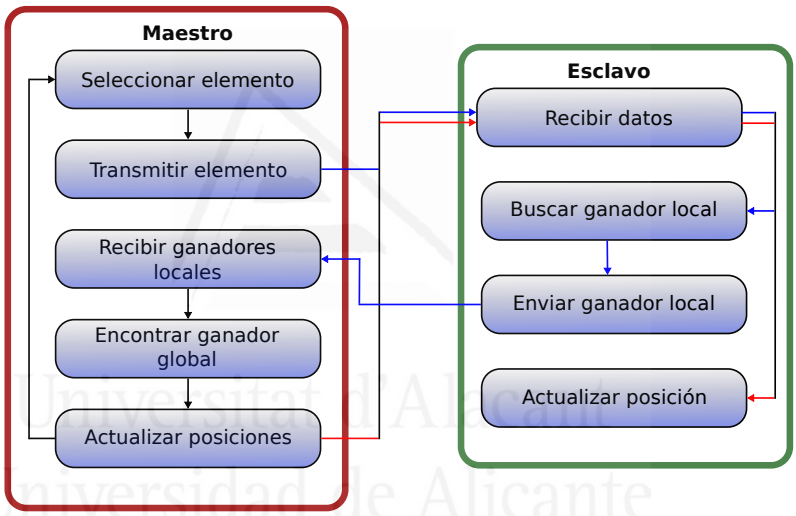


Figura 2.4 Diagrama de una implementación en paralelo de SOM.

Entre los diseños propuestos destaca por su sencillez el que se muestra en la figura 2.4. En este, las neuronas del mapa son divididas entre los nodos disponibles (unidades de procesamiento). En el nodo maestro se seleccionan las entradas y son transmitidas a los nodos esclavos, y espera a que cada uno de estos devuelva los ganadores locales (o sea, las neuronas más similares a la entrada). Luego, se determina el ganador

global y se le notifica al resto de los nodos. Estos, en fase de entrenamiento, determinan si se deben actualizar sus pesos en dependencia de la neurona que resultó la ganadora globalmente.

2.4 Sumario

En este capítulo se abarcan algunos de los temas que conforman la base de la propuesta de solución que se brinda en esta investigación.

Primeramente se abordan algunas de las alternativas existentes orientadas a tratar con grandes volúmenes de datos de manera general, concluyendo con la introducción de la idea de recuperación basada en *clusters*. Esta idea constituye un elemento importante sobre el cual se sustenta la propuesta de solución.

Las generalidades sobre el tema de *clustering* son expuestas de forma somera, y luego son tratados en más detalle las particularidades de las técnicas de *clustering* sobre grandes volúmenes de datos: aspectos relevantes y principales ejemplos. Se dedica un apartado a los métodos de *clustering* con preservación de la topología, ya que luego del análisis del resto de las técnicas se determinó el empleo de un método de este tipo como parte de la propuesta de solución. También se introducen particularidades sobre el empleo de algoritmos de *clustering* para procesar conjuntos de datos compuestos por grafos.

Por último, se incluye un epígrafe que trata el tema de la minería de datos en paralelo. Aquí se abordan aspectos básicos como las arquitecturas paralelas y los tipos de paralelismo, y se exponen ejemplos de algoritmos paralelos de *clustering*, de donde se extrajeron ideas utilizadas en la investigación.

Capítulo 3

Consideraciones de implementación y modificaciones al algoritmo GNG

En este capítulo se presenta el trabajo relacionado con la implementación y las modificaciones realizadas al algoritmo *Growing Neural Gas* (GNG).

La implementación del algoritmo es abordada desde dos perspectivas. Por una parte, la utilización de una estructura de datos que permite disminuir el número de operaciones de cálculo de distancia; y por otra parte, la implementación en paralelo del algoritmo, que permite aprovechar la capacidad de cómputo de arquitecturas multihilo, multinúcleo o *clusters* de computadoras.

Las modificaciones realizadas al algoritmo GNG están orientadas a evitar o atenuar el efecto de la "proliferación de nodos", así como facilitar su uso mediante la incorporación de parámetros que permitan ajustar la estructura del modelo que se desea obtener.

3.1 Algoritmo GNG

En el proceso de diseño de la propuesta de solución de esta investigación se determinó el empleo del algoritmo *Growing Neural Gas* (GNG) en la etapa inicial, correspondiente a la organización del conjunto de datos. El algoritmo GNG es un método capaz de aprender las relaciones topológicas en un conjunto de entrada mediante reglas de aprendizaje de tipo Hebbiano. La idea principal del método es añadir nuevas unidades a una red inicialmente pequeña, mediante la evaluación de medidas estadísticas locales recolectadas durante las etapas previas de adaptación. La descripción detallada del método se muestra en el algoritmo 3.1.

El modelo que se desarrolla como resultado del funcionamiento del algoritmo es una red o grafo formada por:

- Un conjunto $\mathcal{V}$ de vértices (o nodos). Cada nodo $v \in \mathcal{V}$ tiene asociado un vector de referencia $w_v \in \mathbb{R}^n$.
- Un conjunto $\mathcal{A}$ de aristas entre pares de nodos. Estas aristas no tienen pesos asociados, y su único propósito es la definición de la estructura topológica.

El movimiento de adaptación hacia las entradas (paso 6.) conduce a un movimiento general de los nodos hacia las regiones del espacio de entrada de donde vienen las señales ($P(\xi) > 0$). La inserción de aristas (paso 7.) entre el nodo más cercano y el segundo más cercano con respecto a una señal de entrada forma parte de la construcción de la estructura topológica.

La eliminación de aristas (paso 8.) es necesaria para deshacerse de aquellas aristas que han perdido su significado topológico por el movimiento de los nodos que conectan. Este proceso se deduce a partir del envejecimiento de las aristas del nodo ganador (paso 4.), combinado con el reforzamiento de aquellas existentes entre el nodo más cercano y el segundo más cercano (paso 7.).

La acumulación del error durante el proceso de adaptación (paso 5.) ayuda a identificar los nodos que se encuentran en áreas del espacio de entrada donde el mapeado de las señales causa el mayor error. Los nuevos nodos son creados en estas regiones para reducir este error.

Algoritmo 3.1 *Growing Neural Gas* (GNG).

1. Crear los dos primeros nodos a y b en las posiciones aleatorias w_a y w_b.
2. Generar una señal de entrada ξ de acuerdo a la función de distribución $P(\xi)$.
3. Encontrar el nodo más cercano s_1 y el segundo nodo más cercano s_2.
4. Incrementar la edad de las aristas del nodo s_1.
5. Actualizar el error asociado al nodo ganador:

$$e_{s_1}^{t+1} = e_{s_1}^{t} + \|w_{s_1} - \xi\|^2$$

6. Mover s_1 y sus nodos vecinos en la dirección de ξ en las fracciones ϵ_b y ϵ_c, respectivamente, de la distancia total:

$$\begin{aligned} w_{s_1}^{t+1} &= w_{s_1}^{t} + \epsilon_b(\xi - w_{s_1}) \\ w_n^{t+1} &= w_n^{t} + \epsilon_c(\xi - w_n) \qquad \forall n, n \in \mathcal{V}_{s_1} \end{aligned}$$

   donde $\mathcal{V}_{s_1}$ es el conjunto de los nodos vecinos directos de s_1.
7. Si existe una arista entre s_1 y s_2, poner la edad de esta en cero. Si esta arista no existe, crearla.
8. Eliminar las aristas con una edad mayor que ed_{max}. Si como resultado quedara un nodo totalmente desconectado, o sea, sin aristas, eliminarlo.
9. Si el número de señales de entrada generadas hasta el momento es múltiplo de λ (regula el número de inserciones de nuevos nodos):
   - Determinar el nodo q con el mayor valor de error acumulado.

Algoritmo 3.1 *Growing Neural Gas* (GNG). (continuación)

9. • Insertar un nuevo nodo v en la posición media entre el nodo q y su vecino q' con el mayor valor de error acumulado:

$$w_v = 0.5(w_q + w_{q'})$$

   • Conectar mediante aristas el nuevo nodo v con q y q', y eliminar la arista existente entre q y q'.

   • Disminuir el error acumulado en los nodos q y q' por la fracción α. Inicializar el error acumulado del nodo v con el nuevo valor del error acumulado de la variable q.

10. Disminuir el error de todas las variables por la fracción α'.
11. Si no se cumple el criterio de parada regresar al paso 2.

3.2 Consideraciones sobre la implementación de GNG

3.2.1 Implementación de GNG con *Slim-Tree*

A partir del análisis del algoritmo GNG, se puede apreciar que el aspecto computacionalmente más costoso es la búsqueda, en cada iteración, de la neurona ganadora, o sea, la etapa en la que se busca el nodo que se encuentra más cercano a un elemento del espacio de entrada.

Una de las variantes más utilizadas en este tipo de problemas es el empleo de los Métodos de Acceso Métrico (MAM). Los MAM se enfocan en la organización de datos para que, en base a un criterio de similitud, se facilite la búsqueda del conjunto de elementos que están cercanos a un elemento de consulta (Chávez et al., 2001).

En las búsquedas por similitud o proximidad, la similitud entre elementos es modelada a través de una función de distancia que satisface la desigualdad triangular y un conjunto de objetos llamado espacio métrico. Así, se puede considerar un conjunto $\mathcal{U}$ que denota el universo de objetos válidos y la función $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ que mide la distancia entre objetos. Se define como espacio métrico al subconjunto $\mathcal{S} \subseteq \mathcal{U}$, de tamaño $n = |\mathcal{S}|$, llamado diccionario o base de datos. $\mathcal{S}$ denota el conjunto

de objetos de búsqueda. La función de distancia d, debe satisfacer las condiciones de: positividad estricta, simetría, reflexividad y desigualdad triangular. Esta última condición es especialmente importante, pues establece los límites de distancias que aún pueden no haberse calculado, generando algoritmos de búsqueda por similitud significativamente más rápidos (Clarkson, 2006).

Existe una gran variedad de métodos MAM y estos pueden ser clasificados, según su funcionamiento, en basados en agrupamiento y basados en pivotes (Chávez et al., 2001). Los métodos basados en agrupamiento particionan el espacio en regiones representadas por un centroide o centro de grupo, para luego poder descartar regiones completas cuando se hace una búsqueda. Los métodos basados en pivotes seleccionan un conjunto de elementos como pivotes y construyen un índice en base a las distancias entre cada elemento y los pivotes.

A continuación se describen algunos de los métodos MAM clásicos:

- *Burkhard-Keller Tree.* La estructura *Burkhard-Keller Tree* (BKT) (Burkhard and Keller, 1973) inicialmente selecciona un elemento arbitrario $p \in \mathcal{U}$ ($\mathcal{U}$ es el conjunto de datos) como la raíz del árbol. Para cada distancia $i > 0$, se define $\mathcal{U}_i = \{u \in \mathcal{U},\, d(u,p) = i\}$ como el conjunto de todos los elementos a distancia i de la raíz p, y para cada $\mathcal{U}_i$ no vacío, se construye un hijo de p (etiquetado i) para luego recursivamente construir el BKT para $\mathcal{U}_i$. Este método es conveniente para funciones de distancia discreta.

- *Vantage-Point Tree.* El árbol *Vantage-Point Tree* (VPT) (Uhlmann, 1991) fue diseñado para funciones de distancia continua. VPT construye recursivamente un árbol binario seleccionando también un elemento arbitrario $p \in \mathcal{U}$ ($\mathcal{U}$ es el conjunto de datos) como raíz y la mediana de todas las distancias, $M = mediana\{d(p,u), u \in \mathcal{U}\}$. Los elementos a distancia menor o igual a M son insertados en el subárbol izquierdo, mientras que los mayores a M son insertados en el subárbol derecho.

- *M-Tree.* La estructura de datos *M-Tree* (Ciaccia et al., 1997) provee capacidades dinámicas (construcción gradual), además de un

reducido número de cálculos de distancia. Esta estructura es un árbol donde se selecciona un conjunto de elementos representativos en cada nodo y el elemento más cercano a cada uno es organizado en el subárbol cuya raíz es el elemento representativo. Cada elemento representativo almacena su radio de cobertura.

Al hacer una consulta, el elemento de consulta es comparado con todos los elementos representativos del nodo y el algoritmo de búsqueda entra recursivamente en los nodos no descartados usando el criterio del radio de cobertura.

La principal diferencia de *M-Tree* con métodos anteriores es el algoritmo de inserción. Los elementos son insertados en el "mejor" subárbol, definido como aquel subárbol que minimice la expansión del radio de cobertura. El elemento es luego agregado al nodo hoja. En caso de desbordamiento se divide el nodo en dos y se lleva un elemento del nodo a un nivel superior, obteniendo una estructura de datos balanceada.

- *Slim-Tree*. El *Slim-Tree* (Traina et al., 2000) es un MAM dinámico y balanceado con una estructura básica similar al *M-Tree*, donde los datos se almacenan en los nodos hoja. *Slim-Tree* crece de abajo a arriba, de las hojas a la raíz, organizando los objetos en un estructura jerárquica que usa un elemento representativo como el centro de cada región que cubre los objetos en el subárbol.

  Las principales características son la introducción de un nuevo algoritmo de división e inserción, así como el algoritmo *Slim-down* (Traina et al., 2000) y una medida de sobreposición llamada *Fat-factor* para la construcción de árboles más rápidos. El algoritmo de división está basado en el MST (Kruskal, 1956), logrando mejores tiempos de ejecución que otros algoritmos de división y sin sacrificar el rendimiento de las consultas. El algoritmo *Slim-down* es introducido para reducir el grado de sobreposición, haciendo al árbol más estrecho y por lo tanto disminuyendo la cantidad de consultas y el tiempo de construcción.

Slim-Tree

El algoritmo *Slim-Tree* fue el que se determinó utilizar para obtener una implementación eficiente del algoritmo GNG. Esta decisión se debe a los buenos resultados que ofrece (Traina et al., 2000).

En el algoritmo de construcción de *Slim-Tree*, cuando se agrega un nuevo objeto, primero se busca el nodo que cubra a este objeto empezando desde la raíz. En caso de no encontrarse un nodo que cumpla con esta condición, se selecciona un subárbol siguiendo uno de tres posibles métodos:

- Selección aleatoria de uno de los nodos.
- Selección del nodo cuyo elemento representativo posea la mínima distancia al objeto.
- Selección del nodo que tenga la mínima ocupación entre los nodos calificados.

Este proceso se aplica recursivamente en todos los niveles del árbol.

Cuando un nodo se desborda, o sea, el número de hijos que presenta es mayor que un valor predeterminado, se crea un nuevo nodo en el mismo nivel y se distribuyen los objetos entre los nodos, insertándose en el nodo padre. Si el nodo raíz se divide, se crea una nueva raíz y el árbol crece un nivel.

Para la división de los nodos también se han propuesto tres algoritmos:

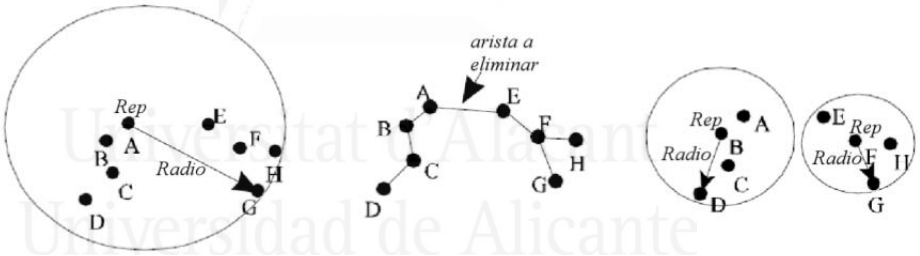
- Aleatorio, donde los dos nuevos centroides son seleccionados aleatoriamente y los objetos existentes son distribuidos entre ellos. Cada objeto es almacenado en el nuevo nodo cuyo centro está más cerca.
- MinMax, donde se consideran todos los pares de objetos como representativos potenciales. Para cada par, se asignan los objetos a un elemento representativo por medio de un algoritmo lineal, escogiendo el par que minimice el radio convergente.

- *Minimal Spanning Tree* (MST) (Kruskal, 1956), donde se construye el MST de los objetos y se desecha el arco más largo del árbol. Gracias al algoritmo de división basado en MST, *Slim-Tree* logra una partición rápida de los nodos en dos grupos que posteriormente reducirán los tiempos de búsqueda.

El algoritmo 3.2 (Traina et al., 2000) describe el funcionamiento de la división de nodos usando MST. El algoritmo considera un grafo de C objetos y $C(C-1)$ aristas, donde el peso de las aristas hace referencia a la distancia entre los objetos conectados. La figura 3.1 muestra de forma gráfica este proceso de división.

Algoritmo 3.2 División mediante MST.

1. Construir el MST de los C objetos.
2. Eliminar la arista más larga.
3. Reportar los objetos conectados como dos grupos.
4. Escoger el objeto representativo de cada grupo, por ejemplo el objeto con la menor distancia máxima a los demás objetos del grupo.



(a) Nodo antes de la división. (b) MST construido con los objetos del nodo. (c) Nodos después de la división.

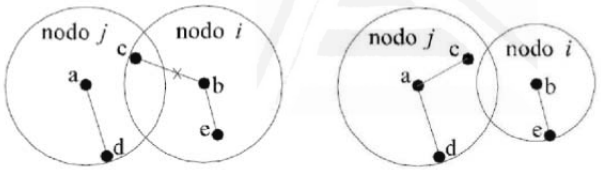
Figura 3.1 División de nodos con el algoritmo MST.

Las regiones de los nodos pueden solaparse unas con otras, incrementando el número de caminos a recorrer al realizar una consulta y, por lo tanto, incrementando el número de cálculos de distancia que se efectúan.

Esta deficiencia se controla mediante la aplicación del algoritmo *Slim-down*. Como resultado, *Slim-Tree* logra producir árboles estrechos y con un reducido grado de solapamiento entre nodos. El algoritmo 3.3 (Traina et al., 2000) describe el proceso del algoritmo *Slim-down* y en la figura 3.2 se muestra un caso ilustrativo del resultado de la aplicación del algoritmo.

Algoritmo 3.3 *Slim-down.*

1. Para cada nodo i en un nivel determinado del árbol, buscar el objeto c más lejano de su centroide.
2. Buscar el nodo j, hermano de i, que también contenga al objeto c; si tal nodo existe, quitar c del nodo i e insertarlo en el nodo j, corrigiendo los radios del nodo i.
3. Repetir los pasos 1 y 2 secuencialmente en todos los nodos de un nivel determinado del árbol. Si en el proceso se produjo un movimiento de objetos de un nodo a otro, repetir los pasos 1 y 2 de nuevo en todos los nodos del nivel.



(a) Nodos antes de aplicar el algoritmo de *Slim-down*.

(b) Nodos después de aplicar el algoritmo de *Slim-down*.

Figura 3.2 Funcionamiento del algoritmo *Slim-down.*

Incorporación del *Slim-Tree* a GNG

La etapa computacionalmente más costosa en el algoritmo GNG es la búsqueda del nodo más cercano a un elemento del espacio de entrada. En el diseño más sencillo, esta búsqueda se realiza de manera secuencial sobre el modelo que conforma el algoritmo y el tiempo de ejecución es

proporcional al número de nodos de este modelo.

Con el objetivo de disminuir el número de operaciones de comparación se incorpora la estructura *Slim-Tree* en la implementación del algoritmo GNG. En esta estructura, los nodos del modelo se almacenan en las hojas, mientras que en el resto de los nodos del *Slim-Tree* se mantiene un elemento representativo correspondiente a un nodo del modelo GNG, y su radio de cobertura, o sea, la distancia mayor entre el elemento representativo y cualquier otro nodo perteneciente a ese subárbol.

En el contexto del algoritmo GNG es necesario modificar el algoritmo *Slim-Tree*, pues las posiciones de los nodos del modelo pueden cambiar. Este cambio de la posición, que ocurre cuando un nodo de GNG es ganador o vecino de un ganador, puede tener implicaciones importantes en la estructura del *Slim-Tree*. Existen dos situaciones donde esto ocurre: cuando el nodo que se modifica es un elemento representativo en uno o varios niveles, o cuando el nodo que se modifica es el más lejano del elemento representativo y, por tanto, define el radio de cobertura de uno o varios nodos del *Slim-Tree*.

La solución para cualquiera de estas dos situaciones es verificar en el nodo hoja si es necesaria una modificación de la estructura *Slim-Tree*, en cuyo caso se propaga la rectificación hacia la raíz del árbol. Si el nodo que se modifica no es el elemento representativo, entonces se comprueba que la distancia de él hasta el nodo representativo no sea superior al radio de cobertura, y esto se realiza en cada nivel hasta llegar a la raíz. Si el elemento que se modifica es un nodo representativo, es necesario buscar un nuevo nodo representativo y propagar esta operación hasta llegar a la raíz del *Slim-Tree*.

Estas operaciones de rectificación no son necesarias en todas las iteraciones del algoritmo GNG, pues mientras mayor sea la cantidad de nodos del modelo, menor va a ser la probabilidad de que el nodo que se modifique altere la estructura del árbol.

3.2.2 Resultados experimentales

En este subepígrafe se muestran los resultados experimentales obtenidos a partir del empleo del *Slim-Tree* en la implementación de GNG.

En la figura 3.3 se muestran los conjuntos de datos que fueron utilizados en la experimentación. Estos conjuntos se asemejan a los utilizados como forma de validación en la literatura sobre este tema. El conjunto *Dataset 1* está formado por puntos ubicados según tres funciones de distribución normal, con distintos valores de desviación estándar. Los conjuntos de datos *Dataset 2* y *Dataset 3* presentan puntos distribuidos de manera uniforme en las regiones que se muestran. Cada conjunto está formado por 10 000 puntos.

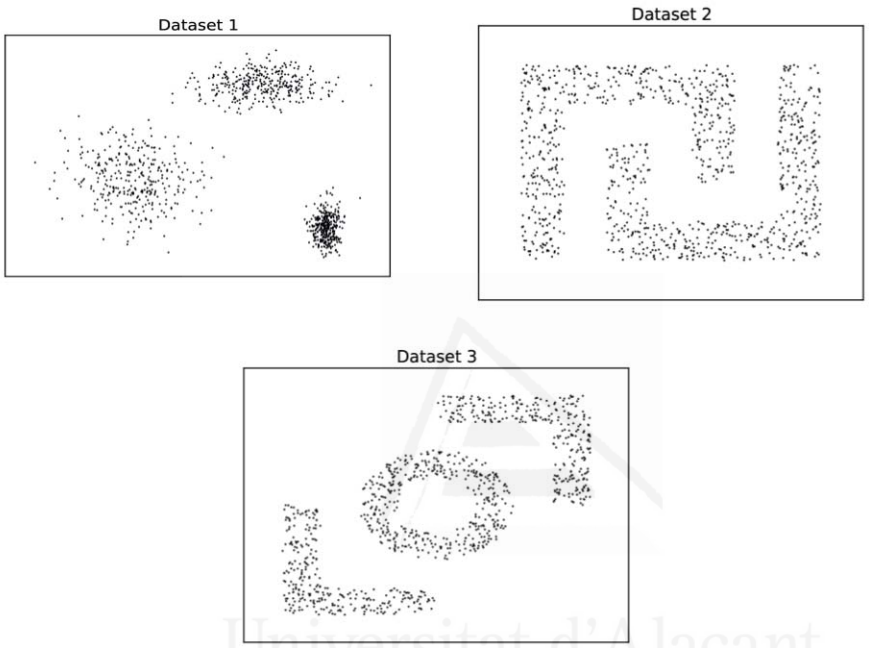
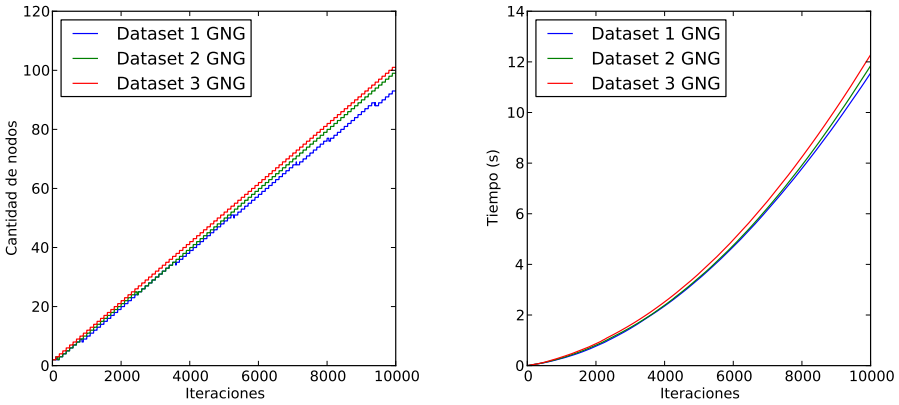


Figura 3.3 Conjuntos de datos de experimentación.

Comportamiento de GNG ante los distintos conjuntos de datos

Primeramente es necesario establecer el comportamiento del algoritmo GNG ante los distintos conjuntos de datos que se utilizan. En la figura 3.4 se aprecia que el comportamiento con respecto a la cantidad de nodos del modelo es muy parecido y, como consecuencia, también el tiempo de ejecución varía muy levemente. Esta relación, como se mencionó anteriormente, se debe a que dentro del algoritmo GNG, la búsqueda del

nodo ganador se realiza entre los nodos del modelo y, por tanto, el coste computacional está directamente relacionado con el tamaño de los modelos, que en todos los casos es similar. En la figura 3.5 se muestran los modelos que se obtienen para cada conjunto de datos.



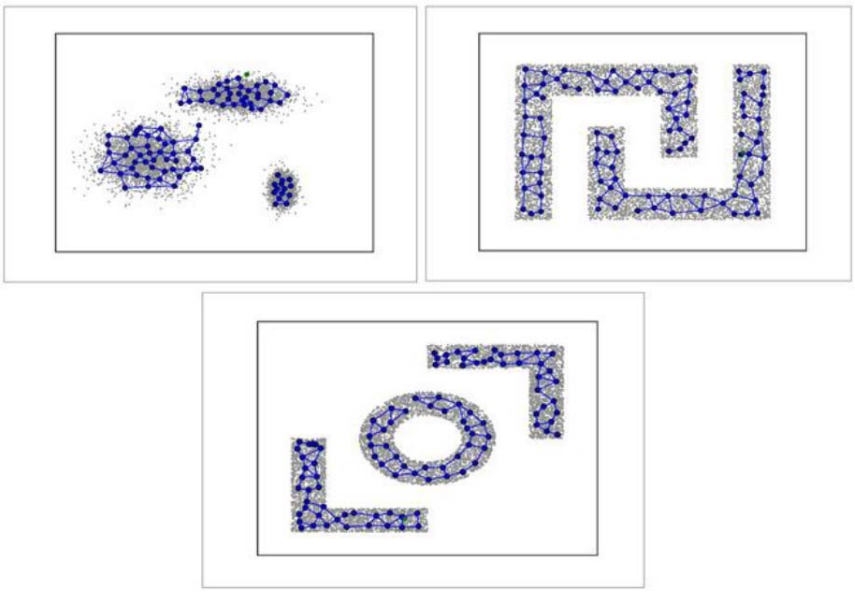
Parámetros GNG ($\lambda = 100$, $\epsilon_b = 0.2$, $\epsilon_c = 0.01$, $ed_{max} = 5$, $\alpha = 0.5$, $\alpha' = 0.9$)

Figura 3.4 Comportamiento del algoritmo GNG ante los distintos conjuntos de datos. Izquierda: Cantidad de nodos del modelo en cada iteración. Derecha: Tiempo de ejecución hasta cada iteración.

Comportamiento de la eficiencia a partir de la utilización de un *Slim-Tree*

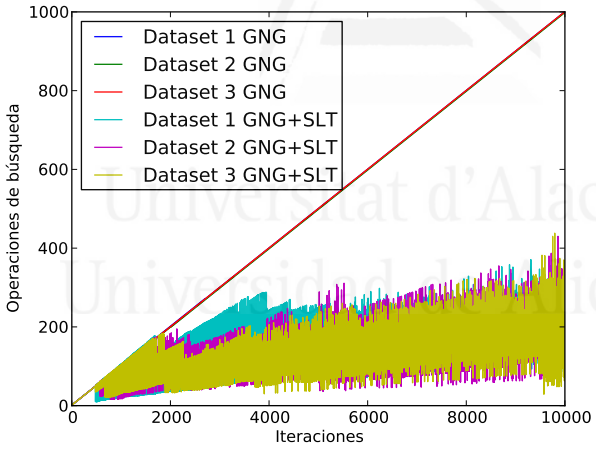
El objetivo de este grupo de experimentos es comprobar la eficiencia de la implementación del algoritmo GNG utilizando la estructura *Slim-Tree*. Para esto se aplicó el algoritmo GNG utilizando la implementación secuencial y con *Slim-Tree* sobre los conjuntos de datos, en condiciones iguales (con los mismos parámetros de GNG).

En la figura 3.6 se compara la cantidad de operaciones de búsqueda que se realizan en la generación del modelo GNG para cada implementación. En este gráfico se puede apreciar que el número de operaciones es lineal con respecto al número de iteraciones para la implementación secuencial, mientras que estos valores están muy por debajo para la implementación que emplea *Slim-Tree*. En este último caso, la cantidad de



Parámetros GNG ($\lambda = 100$, $\epsilon_b = 0.2$, $\epsilon_c = 0.01$, $ed_{max} = 5$, $\alpha = 0.5$, $\alpha' = 0.9$)

Figura 3.5 Modelos generados para los conjuntos de datos por el algoritmo GNG.



Parámetros GNG ($\lambda = 10$, $\epsilon_b = 0.2$, $\epsilon_c = 0.01$, $ed_{max} = 5$, $\alpha = 0.5$, $\alpha' = 0.9$)

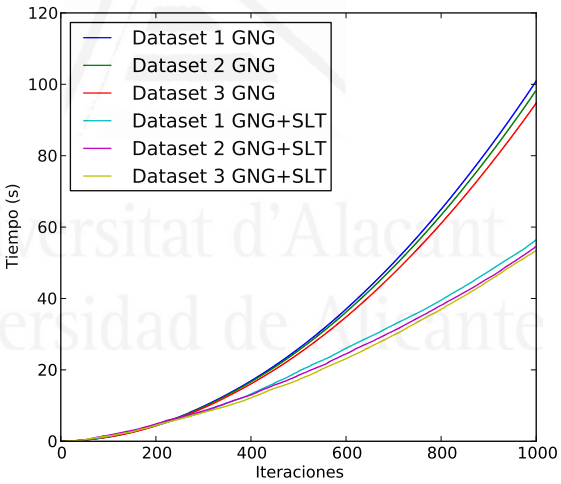
Figura 3.6 Comparación de la cantidad de operaciones de búsqueda que se realizan entre una implementación convencional del algoritmo GNG (GNG) y una implementación utilizando *Slim-Tree* (GNG+SLT).

operaciones varía en cierto rango. Esta variabilidad se debe a que no todas las búsquedas realizan un mismo recorrido en el árbol *Slim-Tree*.

En la tabla 3.1 se muestran los totales del número de operaciones de búsqueda realizadas por cada una de las implementaciones para cada conjunto de datos. En la tabla se aprecia que la mejora en eficiencia con la utilización de *Slim-Tree* es de alrededor del 70 %.

Tabla 3.1 Comparación de la cantidad de operaciones de búsqueda para las implementaciones de GNG: secuencial y utilizando *Slim-Tree*.

	Número de operaciones		
	Dataset 1	*Dataset 2*	*Dataset 3*
GNG convecional	5 007 704	4 988 512	5 006 608
GNG *Slim-Tree*	1 488 791	1 330 540	1 337 466
Mejora (%)	70.72 %	73.32 %	73.28 %



Parámetros GNG ($\lambda = 10$, $\epsilon_b = 0.2$, $\epsilon_c = 0.01$, $ed_{max} = 5$, $\alpha = 0.5$, $\alpha' = 0.9$)

Figura 3.7 Comparación del tiempo de ejecución entre una implementación convencional del algoritmo GNG (GNG) y una implementación utilizando *Slim-Tree* (GNG+SLT).

Finalmente, en la figura 3.7 se observa que, como consecuencia de la disminución en el número de operaciones de búsqueda, los tiempos de ejecución del algoritmo GNG se reducen en casi un 50 %. El tiempo de ejecución, aunque es menor para el caso de la implementación de GNG con *Slim-Tree*, no mejora en la misma proporción que la cantidad de operaciones, pues parte del tiempo se consume en realizar operaciones sobre el *Slim-Tree* (división, *Slim-Down*, rectificación del árbol).

Los resultados mostrados corroboran la mejora en eficiencia que se obtiene con el uso de la implementación descrita del algoritmo GNG con la estructura de datos *Slim-Tree*.

3.2.3 Implementación en paralelo de GNG

Uno de los mecanismos más efectivos para disminuir el tiempo de ejecución en tareas de procesamiento de datos, en particular en minería de datos, es el uso del paralelismo. La implementación de forma paralela de este tipo de aplicaciones, propicia un aprovechamiento eficaz de la capacidad de cómputo disponible en arquitecturas multihilo, multinúcleo o *clusters* de computadoras.

Las redes neuronales en general, y GNG en particular, presentan un diseño en su funcionamiento que las hace especialmente apropiadas para ser implementadas de manera paralela. Esta paralelización se propicia por la posibilidad de realizar de forma simultánea el cálculo de la distancia de un elemento dado a cada uno de los nodos del modelo.

La arquitectura paralela sobre la que se basa la propuesta que se presenta sigue el esquema constituido por un elemento maestro y un grupo de esclavos (figura 3.8). El maestro controla de manera general el funcionamiento del algoritmo, y desde él se le indica a los esclavos lo que deben hacer, mientras que en los esclavos se realiza una parte significativa del procesamiento del algoritmo.

En el maestro es donde se ejecuta el ciclo principal del algoritmo GNG. Aquí se seleccionan los elementos del conjunto de datos que van a ser procesados, y esta información se transmite hacia los esclavos. Posteriormente, a partir de la información que le envían los esclavos, en el maestro se determina el ganador global. El maestro también le indica

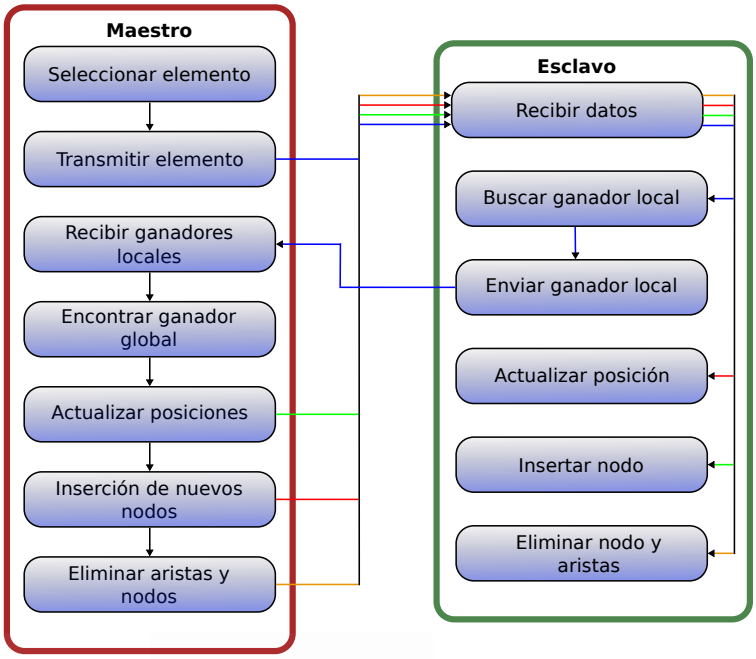


Figura 3.8 Diagrama general del algoritmo GNG en paralelo.

a los esclavos otras operaciones que estos deben realizar: actualizar la posición de nodos, añadir nuevos nodos y eliminar nodos y aristas.

En los esclavos se mantiene el modelo de la GNG de manera distribuida. Cada esclavo mantiene un grupo de nodos sobre los cuales realiza las operaciones que le indica el maestro. La operación fundamental es la búsqueda del nodo que se encuentra más cercano al elemento indicado por el maestro. Cuando un esclavo encuentra el nodo más cercano, se lo notifica al maestro. Una vez que el maestro tiene todos los ganadores locales, es capaz de obtener el ganador global y continuar con el flujo del algoritmo GNG. Esta etapa es la que más influye en el tiempo de ejecución del algoritmo, por este motivo, el maestro trata de mantener una cantidad similar de nodos en cada esclavo. Este balance en la cantidad de nodos se logra insertando nodos en aquellos esclavos con menor cantidad de nodos (ver figura 3.9).

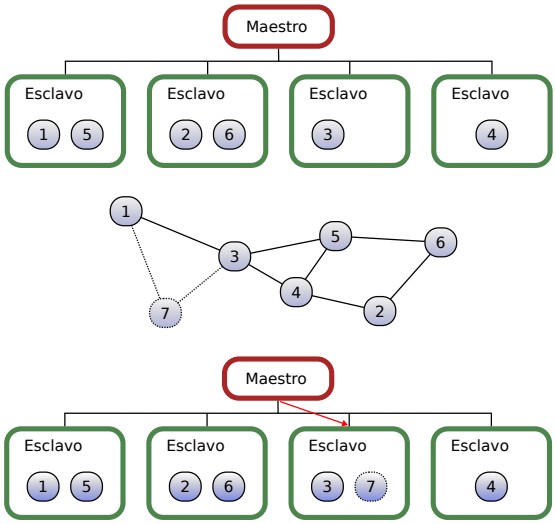


Figura 3.9 Inserción de un nuevo nodo del modelo GNG en uno de los esclavos de la arquitectura.

Solución híbrida

La propuesta final en este trabajo es una solución híbrida en la que se aprovecha la utilidad de cada una de las soluciones individuales sobre contextos diferentes. Por una parte se obtienen los beneficios de la implementación en paralelo, al aprovechar la capacidad de cómputo disponible en varias unidades de procesamiento. Por otra parte, en cada unidad de procesamiento se aprovechan las ventajas del empleo de la estructura *Slim-Tree*.

En la figura 3.10 se muestra un diagrama de la solución híbrida. Aquí se muestra cómo, sobre la base de contar con una implementación en paralelo, se empleará en cada unidad de procesamiento un *Slim-Tree* para organizar los nodos.

La solución híbrida puede aportar niveles de eficiencia superiores a las individuales porque el tiempo de ejecución de la búsqueda del nodo más cercano a un elemento dado se puede reducir con respecto a la solución en paralelo, tanto como se reduce la solución que emplea un *Slim-Tree* con respecto a GNG secuencial.

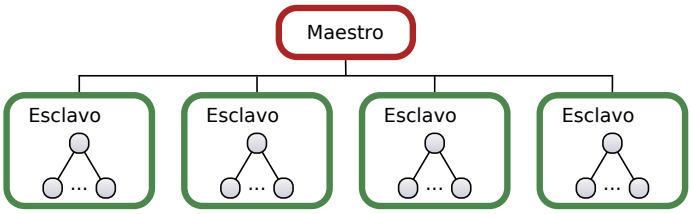
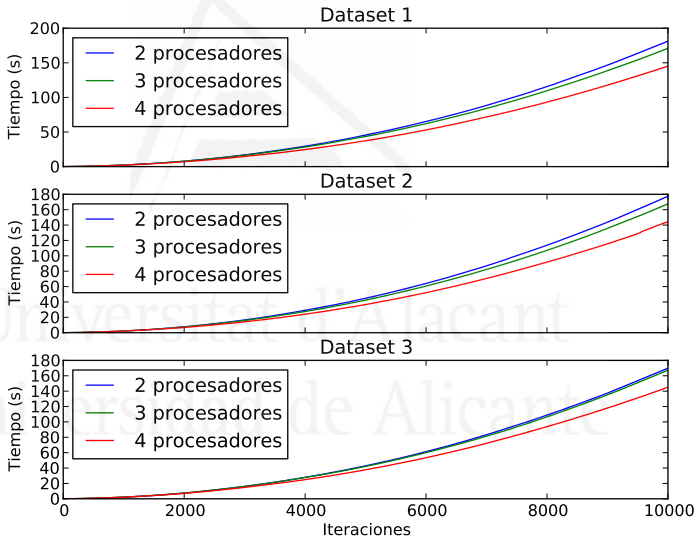


Figura 3.10 Esquema de la solución híbrida, donde se utiliza un *Slim-Tree* en cada esclavo.

3.2.4 Resultados experimentales

Este grupo de experimentos está orientado a comparar la eficiencia de todas las implementaciones del algoritmo GNG descritas en este epígrafe. Con este objetivo se aplicó cada implementación sobre los conjuntos de datos descritos anteriormente, en condiciones similares (con los mismos parámetros del algoritmo GNG).



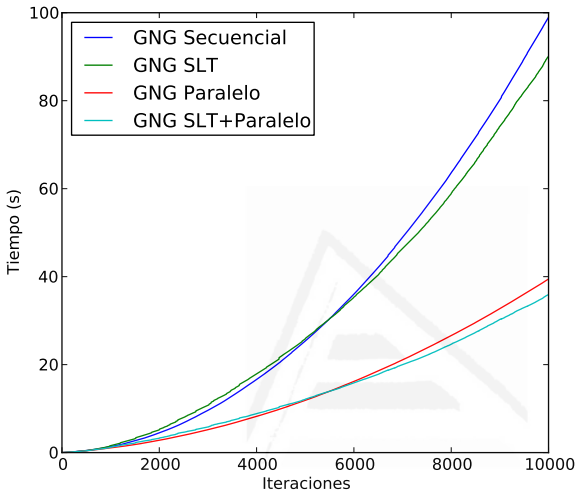
Parámetros GNG ($\lambda = 5$, $\epsilon_b = 0.2$, $\epsilon_c = 0.01$, $ed_{max} = 50$, $\alpha = 0.5$, $\alpha' = 0.9$)

Figura 3.11 Tiempo de ejecución de la implementación en paralelo del algoritmo GNG empleando distinta cantidad de procesadores.

Los resultados específicos de la paralelización del algoritmo GNG se

pueden apreciar en la figura 3.11. Aquí se observa cómo, según se emplean más procesadores, disminuye el tiempo de ejecución del algoritmo, evidenciando la escalabilidad de esta implementación.

En la figura 3.12 se muestra una comparación final entre el tiempo de ejecución del algoritmo GNG, con las implementaciones consideradas: la secuencial, empleando un *Slim-Tree*, en paralelo e híbrida. Aquí se puede apreciar que el mejor resultado lo obtiene la implementación final propuesta (híbrida), que combina la implementación en paralelo con el uso del *Slim-Tree* en cada proceso paralelo.



Parámetros GNG ($\lambda = 20$, $\epsilon_b = 0.2$, $\epsilon_c = 0.01$, $ed_{max} = 5$, $\alpha = 0.5$, $\alpha' = 0.9$)
Las soluciones en paralelo (GNG Paralelo y GNG SLT+Paralelo) utilizan cuatro procesadores

Figura 3.12 Comparación del tiempo de ejecución de las implementaciones: secuencial, empleando *Slim-Tree*, en paralelo e híbrida del algoritmo GNG.

3.3 Modificaciones al algoritmo GNG

En este epígrafe se exponen las modificaciones que se le han incorporado al algoritmo GNG con el objetivo de facilitar y adecuar su utilización

como parte de la propuesta de solución que se elabora en esta investigación. Por una parte se intenta reducir la importancia de la elección de algunos parámetros del algoritmo, mientras que por otra parte se trata de evitar la proliferación de nodos en regiones muy densas del conjunto de datos de entrada.

3.3.1 Inserción de nodos en regiones no visitadas

Una idea empleada en algunos algoritmos de agrupamiento (Carpenter and Grossberg, 2003; Furao and Hasegawa, 2007; Hebboul et al., 2011) es la inserción de un nuevo nodo si este se encuentra en una región que, de alguna manera, resulta desconocida en ese momento del aprendizaje. En el algoritmo GNG, el aprendizaje se produce por el movimiento de los nodos en dependencia del conjunto de datos y por la inserción de nuevos nodos cada cierto número de iteraciones. Debido a esto, el proceso de aprendizaje de nuevas regiones en el algoritmo GNG se ralentiza, al no contar con un mecanismo que permita la inserción de nuevos nodos en estas regiones de manera inmediata.

Teniendo en cuenta las ventajas que ofrece, se decidió incorporar esta idea al algoritmo GNG. Esta variante acelera la velocidad de crecimiento del modelo, sobre todo en las primeras etapas, y como resultado acelera también el proceso de aprendizaje.

La incorporación de nodos en regiones nuevas requiere que cada nodo del modelo tenga asociado un umbral que refleja la región que abarca. Cuando se analiza un nuevo elemento del conjunto de datos, si el elemento no está dentro del umbral de alguno de los dos nodos de los cuales se encuentra más cercano, se crea un nuevo nodo en la posición de dicho elemento, y se crea una arista entre él y el nodo más cercano. El umbral T de un nodo i está definido como la distancia entre él y su vecino más lejano:

$$T_i = \max_{v \in \mathcal{V}_i} \|w_v - w_i\| \tag{3.1}$$

donde $\mathcal{V}_i$ es el conjunto de nodos para los cuales existe una arista entre i y v.

En cada iteración del algoritmo GNG, ocurre el movimiento de un

grupo de nodos (el ganador y sus vecinos). Estos movimientos pueden traer como consecuencia que los valores del umbral para algunos nodos estén desactualizados y, por tanto, es necesario llevar a cabo la actualización del umbral. Esta actualización no es demasiado costosa computacionalmente, pues solo es necesario analizar una región local del modelo.

La región que se debe actualizar está formada por el nodo ganador, sus vecinos y los vecinos de sus vecinos. El nodo ganador debe ser actualizado, pues al cambiar de posición, su umbral se modifica, ya que este está determinado por alguno de sus vecinos, que también se movieron y no en la misma magnitud, y probablemente, tampoco en el mismo sentido. En el caso de los nodos vecinos del ganador, también se mueven de posición, y tienen un vecino que se mueve más que él (el nodo ganador); además pueden tener vecinos que se mueven igual que él (en magnitud, pero muy poco probablemente en sentido), y vecinos que no se mueven, así que también se deben actualizar. Por último, los nodos que se encuentran a dos saltos del nodo ganador, aunque no se mueven, si lo hacen un grupo de sus vecinos, entre los que pudieran estar los que determinan su umbral, por tanto, también deben ser actualizados.

3.3.2 Eliminación de los parámetros de adaptación de los nodos ganadores

La actualización de la posición del nodo ganador y sus vecinos se lleva a cabo, en el algoritmo GNG, mediante el movimiento de estos nodos en la dirección del elemento del conjunto de entrada que se analiza en cada iteración. La magnitud de este movimiento está regulada por los parámetros ϵ_b y ϵ_c ($\epsilon_b > \epsilon_c$) para el ganador y sus vecinos, respectivamente. En el algoritmo GNG estos parámetros se mantienen constantes durante todo el procesamiento del conjunto de datos.

Esta forma de funcionamiento presenta dos desventajas fundamentales. En primer lugar, el usuario que utiliza el algoritmo debe definir estos valores, y aunque existen valores empleados con éxito (Fritzke, 1995), el comportamiento puede variar en dependencia del conjunto de datos. En segundo lugar, esta variante no considera la evolución del modelo

en la medida que se desarrolla, más bien está ideada para funcionar de manera adecuada ante conjuntos de datos no estacionarios.

Una idea empleada en otros algoritmos de aprendizaje (Kohonen, 2001) es utilizar un esquema similar al enfriamiento de algoritmos heurísticos, que permite una adaptación global en las primeras etapas del aprendizaje y un ajuste fino hacia el final. En (Furao and Hasegawa, 2007) se propone un proceso de adaptación de los nodos ganadores que elimina la necesidad de establecer los parámetros ϵ_b y ϵ_c. Este mecanismo se basa en la idea de que aquellos nodos que resultan ganadores una mayor cantidad de veces presentan una ubicación favorable en el espacio de los elementos del conjunto de datos y, por tanto, el grado de adaptación de estos ante nuevas entradas debe ser menor para no alejarse de esta localización. A partir de esta idea, la actualización del nodo ganador y sus vecinos queda como:

$$w_{s_1}^{t+1} = w_{s_1}^{t} + \frac{1}{wins_{s_1}}(\xi - w_{s_1}) \tag{3.2}$$

$$w_{n}^{t+1} = w_{n}^{t} + \frac{1}{100 \times wins_{n}}(\xi - w_{n}) \qquad \forall n, n \in \mathcal{V}_{s_1} \tag{3.3}$$

donde w_{s_1} es el vector de referencia del nodo ganador s_1, $\mathcal{V}_{s_1}$ es el conjunto de nodos vecinos del nodo s_1, $wins_i$ es la cantidad de veces que el nodo i ha resultado ganador y ξ es el elemento del conjunto de datos que se está procesando.

3.3.3 Restricciones en la etapa de inserción de nuevos nodos

Uno de los comportamientos indeseados en los modelos generados por el algoritmo GNG, es la proliferación de nodos en determinadas regiones del conjunto de datos que presentan una alta densidad (Satizábal et al., 2008). Esta proliferación de nodos, se produce debido a la ausencia de un criterio para detener la inserción de nodos en áreas suficientemente representadas. La proliferación de nodos trae como consecuencia negativa un modelo desequilibrado, con regiones más amplias menos representadas; y además, el crecimiento del modelo conlleva un aumento en el tiempo de ejecución del algoritmo.

En este trabajo se presentan tres restricciones que se aplican al proceso de inserción de nuevos nodos del algoritmo GNG. Cada una de las restricciones tiene en cuenta un criterio distinto que intenta mantener cierto grado de homogeneidad en el modelo que se conforma. Esto se logra introduciendo un nuevo parámetro para cada restricción, que controla la razón entre una medida local o temporal utilizada para el criterio y la medida máxima a nivel global.

El empleo de estas restricciones le brinda robustez al algoritmo GNG con respecto al parámetro λ, que es el que regula cuántas inserciones de nuevos nodos se realizan. Este parámetro tiene una marcada influencia en la cantidad de nodos del modelo resultante, pues aunque GNG tiene un mecanismo de eliminación de nodos, el efecto de este en el tamaño del modelo es bastante reducido. Por tanto, a partir de la aplicación de las restricciones, la cantidad de nodos va a estar más vinculada a los parámetros asociados a las restricciones que al parámetro λ. Estos parámetros tienen una interpretación más relacionada con el resultado que se desea obtener pues, además de estar diseñados para presentar valores entre 0 y 1, tienen un reflejo en el tipo de estructura del modelo resultante.

Restricción basada en el radio de los nodos

Una medida que puede reflejar el grado representatividad que existe en una región de un modelo es el radio de los nodos, definido como la distancia promedio de un nodo a cada uno de sus vecinos:

$$r_q = \frac{1}{|\mathcal{V}_q|} \sum_{v \in \mathcal{V}_q} \|w_q - w_v\| \tag{3.4}$$

donde $|\mathcal{V}_q|$ es la cantidad de elementos del conjunto de vecinos del nodo q.

A partir de esta definición, se desea obtener un modelo donde el radio de los nodos se encuentre dentro de un rango de valores, produciendo un modelo con un enrejado relativamente uniforme. El mecanismo que se utiliza para lograr esta homogeneidad es evitar la inserción de nuevos nodos en regiones donde el radio de un nodo es desproporcionadamente

inferior con respecto al nodo de mayor radio del modelo. Así, se realiza una inserción alrededor del nodo q si se cumple que:

$$\frac{r_q}{r_{max}} > \varphi \tag{3.5}$$

donde r_q es el radio del nodo q, r_{max} es el radio del nodo de mayor radio del modelo y φ es el parámetro que se añade al algoritmo para regular el proceso de inserción de nuevos nodos.

Restricción basada en la densidad

La densidad de un nodo es una medida que ha sido utilizada para contrarrestar la proliferación de nodos (Furao and Hasegawa, 2007), evitando insertar nuevos nodos en regiones con una alta densidad. En general, la densidad de un nodo es definida a partir de la cantidad de elementos del conjunto de datos acumulados alrededor de un nodo. Así, si muchos elementos se encuentran cerca de un nodo este posee una alta densidad, y una baja densidad en el caso contrario. Algunos algoritmos como SOINN (Furao and Hasegawa, 2006) estiman la densidad de un nodo teniendo en cuenta la cantidad de veces que un nodo ha sido ganador. Esta variante presenta, sin embargo, los siguientes problemas:

- Existen numerosos nodos que están en las áreas de alta densidad. Como consecuencia, la probabilidad de que un nodo sea ganador en estas áreas no es considerablemente más alta que la de los nodos ubicados en áreas de baja densidad, dado que hay muchos candidatos a ser ganador.
- En los métodos de aprendizaje incremental algunos nodos generados en las primeras etapas no serán ganadores por un largo tiempo. En estos casos, estos nodos pueden ser considerados de baja densidad en etapas posteriores.

En (Furao and Hasegawa, 2007) se introduce una definición de densidad que tiene en cuenta, además de las veces que un nodo ha sido ganador, la relación de un nodo con sus vecinos. Para calcular la densidad de un nodo primeramente se calcula el radio de un nodo i (r_i) como

en la ecuación 3.4. A partir de este radio se define el *punto* de un nodo q como:

$$p_i = \begin{cases} \frac{1}{(1+r_i)^2} & \text{si el nodo } i \text{ es ganador} \\ 0 & \text{si el nodo } i \text{ no es ganador} \end{cases} \tag{3.6}$$

De la definición de punto se puede concluir que si la distancia media del nodo i a sus vecinos (r_i) es grande, entonces el número de nodos en esa área es pequeño, y por consiguiente, la distribución de nodos es dispersa y la densidad es baja. Por el contrario, si r_i es pequeño, esto significa que el número de nodos en esa área es alto, obteniendo un mayor valor de p_i.

Los valores del *punto* de un nodo solo se calculan cuando el nodo es el ganador, por lo que en una iteración, los *puntos* del ganador cambian, pero los de los restantes permanecen iguales.

Los puntos acumulados a_i se calculan mediante la suma de los puntos para el nodo i durante un periodo de aprendizaje:

$$a_i = \sum_{k=1}^{n} \left(\sum_{l=1}^{\lambda} p_i \right) \tag{3.7}$$

donde n indica la cantidad de periodos de aprendizaje, y se calcula como:

$$n = \frac{LT}{\lambda} \tag{3.8}$$

donde LT es el número total de elementos del conjunto de entradas.

Finalmente, la densidad del nodo i se calcula como:

$$D_i = \frac{1}{N} a_i \tag{3.9}$$

donde N representa la cantidad de periodos donde a_i es mayor que 0.

Teniendo en cuenta esta definición de densidad, la idea es intentar mantener un nivel de homogeneidad a nivel de densidad. Para esto, en el proceso de inserción de nuevos nodos se incorpora la restricción:

$$\frac{D_q}{D_{max}} > \gamma \tag{3.10}$$

donde D_q es la densidad del nodo q, D_{max} es la densidad del nodo de mayor densidad en el modelo y γ es el parámetro que regula la diferencia de densidad máxima permitida, a partir de la cual se restringe la inserción de nuevos nodos en el modelo.

Restricción basada en el seguimiento del error cuadrático

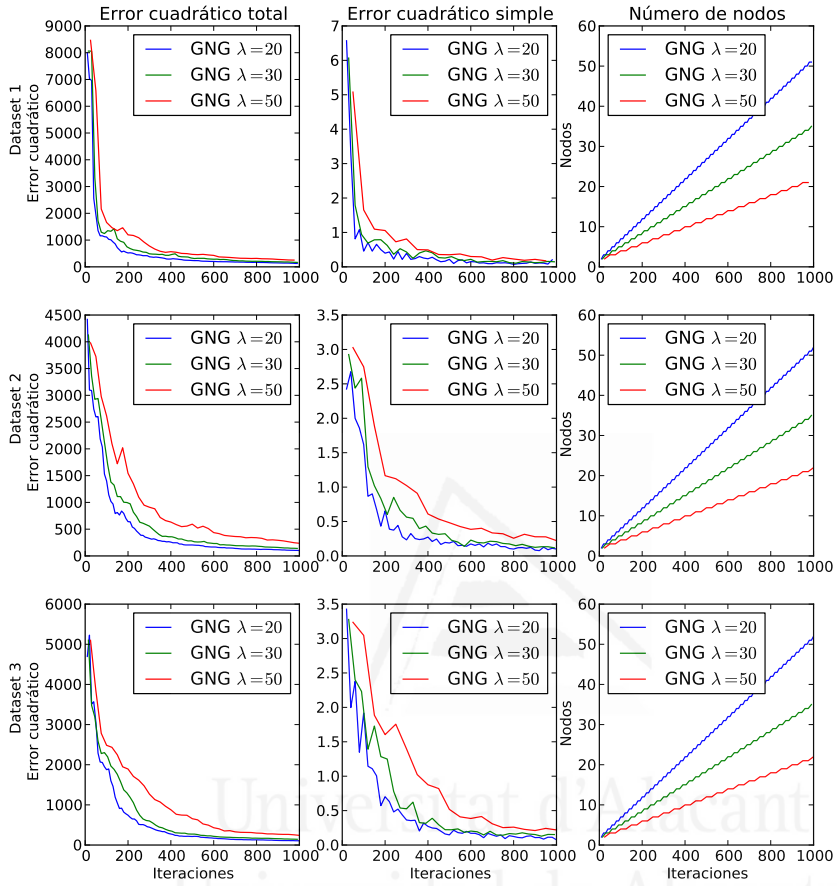
Un aspecto que contribuye a la proliferación de nodos es que el algoritmo GNG no cuenta con un criterio que detenga el crecimiento del modelo basado en la calidad del mismo. Una de las medidas de calidad aplicables a los algoritmos de *clustering* es el error cuadrático e en la clasificación de todos los elementos del conjunto, o sea:

$$e = \sum_{\xi \in \mathcal{D}} \min_{v \in \mathcal{V}} \|\xi - v\|^2 \tag{3.11}$$

donde $\mathcal{D}$ es el conjunto de entrada y $\mathcal{V}$ el conjunto de nodos del modelo. Así, una idea para detener el crecimiento del modelo es continuar el crecimiento hasta alcanzar cierto valor de calidad.

En la figura 3.13 se puede apreciar el comportamiento del error cuadrático (*error cuadrático total* en la figura) para distintos valores del parámetro λ, utilizando cada uno de los conjuntos de datos empleados en los experimentos del epígrafe anterior. Aquí, se observa cómo el valor del error cuadrático disminuye a medida que se desarrolla el proceso de aprendizaje. En general, las curvas correspondientes a valores de λ mayores se encuentran por encima de las curvas correspondientes a valores de λ menores. Esto se debe a que mientras mayor es el valor de λ, más rápidamente crece el modelo y, al tener más nodos, los modelos logran una mayor representatividad y, por tanto, menor valor del error cuadrático.

También es necesario señalar que en todos los casos, tras una etapa de aprendizaje acelerado con un decrecimiento importante en el valor del error cuadrático, se alcanza una cierta estabilidad. Esto significa que a partir de cierto momento, un crecimiento en el número de nodos no produce un descenso significativo en el error cuadrático. De hecho, se llega a un punto en que tampoco existe una diferencia significativa entre las curvas correspondientes a distintos valores de λ.



Parámetros GNG ($\epsilon_b = 0.2$, $\epsilon_c = 0.01$, $ed_{max} = 50$, $\alpha = 0.5$, $\alpha' = 0.9$)

Figura 3.13 Comportamiento del error cuadrático total, el error cuadrático simple y el número de nodos, para distintos valores del parámetro λ.

A partir de este análisis, se puede plantear un requisito de inserción análogo a los criterios anteriores, estableciendo que para realizar la inserción de un nodo en el modelo la razón entre el error cuadrático en el momento de la inserción (e_j, error cuadrático en la iteración j) y el error cuadrático mayor alcanzado durante el aprendizaje (e_{max}) sea mayor que un nuevo parámetro, σ, que se incorpora al algoritmo:

$$\frac{e_j}{e_{max}} > \sigma \tag{3.12}$$

Esta propuesta tiene, sin embargo, un inconveniente importante: el cálculo del error cuadrático es sumamente costoso computacionalmente, pues se debe recorrer todo el conjunto de datos. Teniendo en cuenta este aspecto invalidante se intentó emplear alguna medida relacionada con el error cuadrático, que no implicase un coste computacional apreciable.

Así, se encontró la existencia de una relación relevante entre el error cuadrático y lo que se define como *error cuadrático simple*, o sea, la distancia del elemento del espacio de entrada al nodo del modelo del cual se encuentra más cercano:

$$\hat{e}_j = \|w_{s_1} - \xi\|^2 \tag{3.13}$$

donde $\hat{e}_j$ es el error cuadrático simple en la iteración j, ξ es el elemento del espacio de entrada y w_{s_1} es el vector de referencia del nodo ganador. Esta relación está sustentada por el hecho de que en la medida que el modelo se amolda al conjunto de datos, tanto por crecimiento como por adaptación, este va a lograr una mayor representatividad del mismo y, en general, los nuevos elementos que se analizan van a estar más cercanos a algún nodo. Sin embargo, está claro que en muchos casos, en especial ante la presencia de ruido o durante el reconocimiento de nuevas regiones, estos valores pueden diferir bastante. Teniendo en cuenta esta posible variabilidad se utilizó el promedio de los valores del error cuadrático simple ($\bar{E}_k$) entre dos momentos de inserción (k y $k+1$), o sea:

$$\bar{E}_k = \frac{1}{\lambda} \sum_{j=k\lambda}^{(k+1)\lambda} \hat{e}_j \tag{3.14}$$

En la tabla 3.2 se muestran los coeficientes de correlación entre el error cuadrático simple entre dos momentos de inserción y el error cuadrático total en una iteración. Los altos valores de correlación apoyan la viabilidad de utilizar el comportamiento del error cuadrático simple entre dos momentos de inserción para estimar el comportamiento del error cuadrático total en una iteración. La ventaja de emplear esta estimación radica en que prácticamente no conlleva una carga computacional extra, pues el cálculo de la distancia entre un elemento del espacio de entrada y el nodo del modelo del cual se encuentra más cercano, forma parte del funcionamiento del algoritmo GNG. Finalmente, la condición que se impondría a la inserción de un nuevo nodo es:

$$\frac{\bar{E}_k}{\bar{E}_{max}} > \sigma \tag{3.15}$$

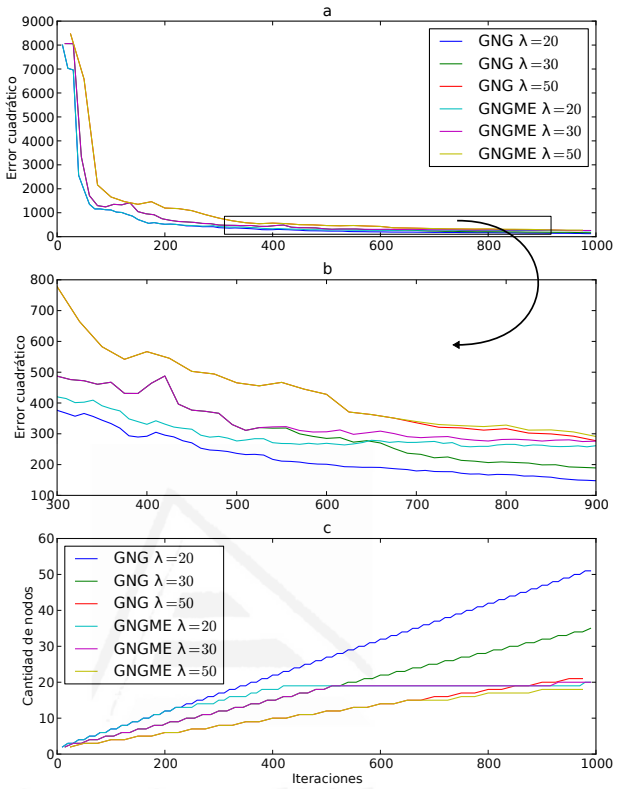
donde σ es el parámetro que se introduce para evitar la inserción cuando el error no ha disminuido significativamente, evitando a su vez, un crecimiento innecesario del modelo.

Tabla 3.2 Correlación entre el error cuadrático total y el error cuadrático simple para los distintos conjuntos de datos.

Conjunto de datos	$\lambda = 20$	$\lambda = 30$	$\lambda = 50$
Dataset 1	0.9841837	0.9936787	0.9953250
Dataset 2	0.9847602	0.9798915	0.9787587
Dataset 3	0.9669757	0.9925371	0.9731519

En la figura 3.14 se muestra una comparación entre los resultados que se obtienen al aplicar el algoritmo GNG con y sin la restricción basada en el seguimiento del error cuadrático. En la figura 3.14a se observa como en las primeras etapas el comportamiento del valor del error cuadrático es el mismo pues no se cumple la restricción, y se forma el mismo modelo. A partir de cierto momento se comienza a cumplir la restricción, lo cual se aprecia en la figura 3.14c, cuando varía el crecimiento en la cantidad de nodos del modelo. En esta figura se puede apreciar además, cómo independientemente del valor de λ, los modelos resultantes que

se obtienen para los algoritmos que incluyen la restricción presentan un tamaño similar, además de un valor del error cuadrático similar.



Parámetros comunes ($\epsilon_b = 0.2$, $\epsilon_c = 0.01, ed_{max} = 50, \alpha = 0.5, \alpha' = 0.9$)
Parámetros GNGME ($\sigma = 0.2$)

Figura 3.14 Comparación del comportamiento del error cuadrático (a y b) y el número de nodos (c) en el algoritmo GNG con (GNGME) y sin la restricción en la inserción basada en seguimiento del error (GNG), para los distintos conjuntos de datos.

En general, se puede concluir que como resultado de la incorporación de esta restricción al algoritmo GNG, se logra cierto grado de robustez con respecto al parámetro λ, pues se obtienen modelos similares en di-

mensiones y calidad.

3.3.4 Algoritmo GNG modificado (GNGM)

Finalmente, el algoritmo propuesto (GNGM), cuya base es GNG, y al que se le han agregado las modificaciones descritas, se muestra en el algoritmo 3.4.

En el marco de esta investigación, el algoritmo propuesto va a ser empleado para procesar un conjunto de datos cuya forma de representación serán grafos. Por este motivo es necesario utilizar una representación vectorial de estos datos, reflejada en el algoritmo como w_ξ para cada grafo ξ.

La inserción de nodos en regiones no visitadas permite acelerar el proceso de aprendizaje. Esto se logra mediante la incorporación de un paso de inserción cuando una nueva entrada no se encuentra dentro del área de cobertura de alguno de los nodos más cercanos a la entrada (paso 4.).

También se ha modificado el proceso de adaptación del nodo ganador y sus vecinos, pues se han eliminado los coeficientes que regulan este movimiento, y se ha sustituido por un esquema dinámico, que depende de la cantidad de veces que un nodo ha resultado ganador (paso 7.).

Finalmente, se han incluido un grupo de condiciones en la etapa de inserción de nuevos nodos (paso 11.). Estas restricciones intentan en su conjunto atenuar el efecto de la proliferación de nodos, además de aumentar la robustez del algoritmo con respecto al parámetro λ. En el algoritmo se observan, además, otros pasos adicionales con respecto al algoritmo GNG, que son pasos complementarios de las modificaciones fundamentales.

En el algoritmo propuesto se han eliminado los parámetros ϵ_b y ϵ_c, relacionados con los movimientos de adaptación del modelo, mientras que se han agregado los parámetros φ, γ y σ, vinculados a las restricciones basadas en el radio de los nodos, la densidad de los nodos y el error cuadrático, respectivamente. Los tres parámetros tienen características comunes: pueden tomar valores entre 0 y 1 y entre más pequeño es su valor menos restrictivos son. Si los tres valores son cero, el paso de inserción de nuevos nodos queda como en el algoritmo GNG original.

Algoritmo 3.4 *Growing Neural Gas* modificado (GNGM).

1. Crear los dos primeros nodos en las posiciones de los dos primeros elementos seleccionados de forma aleatoria del conjunto de datos.
2. Seleccionar un elemento aleatorio ξ del conjunto de datos.
3. Encontrar el nodo más cercano s_1 y el segundo nodo más cercano s_2, y actualizar la cantidad de victorias $wins_{s_1}$ del nodo s_1.
4. Si el vector de referencia del elemento seleccionado no se encuentra dentro del umbral de los nodos más cercanos ($\|w_{s_1} - w_\xi\| > T_{s_1}$ y $\|w_{s_2} - w_\xi\| > T_{s_2}$):
   - Insertar un nuevo nodo v_ξ en la posición w_ξ.
   - Crear una arista entre s_1 y v_ξ.
   - Actualizar el umbral T, el radio r y la densidad D para los nodos s_1 y v_ξ.
   - Volver al paso 2.
5. Incrementar la edad de las aristas del nodo s_1.
6. Actualizar el error asociado al nodo ganador y error cuadrático entre dos momentos de inserción:

$$e_{s_1}^{t+1} = e_{s_1}^{t} + \|w_{s_1} - w_\xi\|^2$$
$$\bar{E}^{t+1} = \bar{E}^{t} + \|w_{s_1} - w_\xi\|^2$$

7. Mover s_1 y sus nodos vecinos en la dirección de ξ:

$$w_{s_1}^{t+1} = w_{s_1}^{t} + \frac{1}{wins_{s_1}}(\xi - w_{s_1})$$
$$w_n^{t+1} = w_n^{t} + \frac{1}{100 \times wins_n}(\xi - w_n) \qquad \forall n, n \in \mathcal{V}_{s_1}$$

donde $\mathcal{V}_{s_1}$ es el conjunto de nodos vecinos del nodo s_1.

Algoritmo 3.4 *Growing Neural Gas* modificado (GNGM). (continuación)

8. Actualizar el umbral T, el radio r y la densidad D para el nodo ganador s_1, sus vecinos directos, y los vecinos directos de estos.
9. Si existe una arista entre s_1 y s_2, poner la edad de esta en cero. Si esta arista no existe, crearla.
10. Eliminar las aristas con una edad mayor que ed_{max}. Si como resultado quedara un nodo totalmente desconectado, eliminarlo.
11. Si el número de elementos procesados hasta el momento es múltiplo de λ y se cumplen las condiciones de inserción:

$$\frac{r_q}{r_{max}} > \varphi \,, \frac{D_q}{D_{max}} > \gamma \,, \frac{\bar{E}}{\bar{E}_{max}} > \sigma$$

- Determinar el nodo q con el mayor valor de error acumulado.
- Insertar una nuevo nodo v en la posición media entre el nodo q y su vecino q' con el mayor valor de error acumulado:

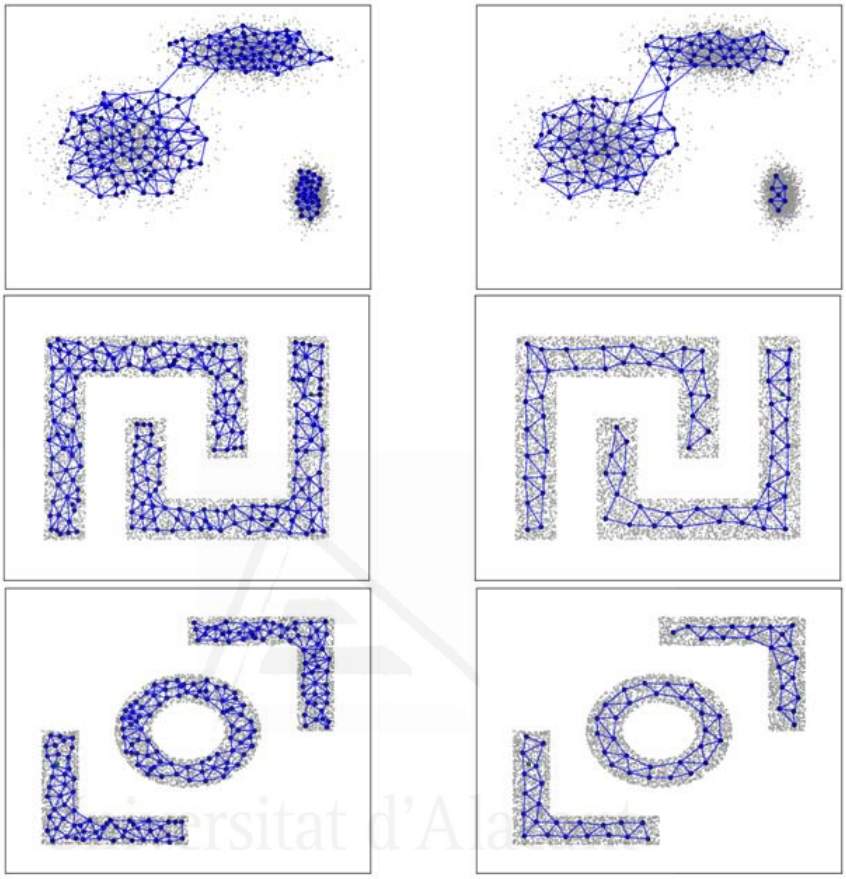
$$w_v = 0.5(w_q + w_{q'})$$

- Conectar mediante aristas el nuevo nodo v con q y q', y eliminar la arista existente entre q y q'.
- Disminuir el error acumulado en los nodos q y q' por la fracción α. Inicializar el error acumulado del nodo v con el nuevo valor del error acumulado de la variable q.

12. Disminuir el error de todas las variables por la fracción α'.
13. Volver al paso 2.

En la figura 3.15 se comparan los resultados que se obtienen cuando se emplean los algoritmos GNG y GNGM. En los experimentos se han utilizado los mismos valores para los parámetros que son comunes a ambos algoritmos, para tratar imponerles condiciones básicas similares.

En los resultados se observa una diferencia evidente en la cantidad de nodos del modelo obtenido por cada algoritmo para cada conjunto de



Parámetros comunes ($\lambda = 50$, $ed_{max} = 50$, $\alpha = 0.5$, $\alpha' = 0.9$)
Parámetros GNG ($\epsilon_b = 0.2$, $\epsilon_c = 0.01$)
Parámetros GNGM ($\varphi = 0.15$, $\gamma = 0.15$, $\sigma = 0.15$)

Figura 3.15 Comparación de los modelos generados por el algoritmo GNG (izquierda) y GNGM (derecha).

datos, siendo mucho menor para los modelos obtenidos con el algoritmo GNGM. Ambos algoritmos tienen la misma cantidad de intentos de inserción de nuevos nodos alrededor del nodo de mayor error acumulado pues los conjuntos son los mismos y el valor de λ también. De hecho, el algoritmo propuesto agrega nodos también como parte del proceso de aprendizaje acelerado. Sin embargo, el resultado es que con el algoritmo GNGM se obtienen modelos de menos nodos (con una calidad representativa apreciable) gracias a las restricciones impuestas en la fase de inserción.

3.4 Sumario

La primera parte de este capítulo se dedica a explicar los aspectos relacionados con una implementación eficiente del algoritmo GNG. La implementación que se utiliza combina el uso de la estructura de datos *Slim-Tree* y una implementación en paralelo del algoritmo. La estructura de datos *Slim-Tree* reduce considerablemente el número de operaciones necesarias para hallar el nodo ganador, mientras que la implementación en paralelo del algoritmo reduce el tiempo de ejecución distribuyendo las operaciones entre distintas unidades de procesamiento que realizan el trabajo de manera simultánea. Los resultados experimentales confirmaron la eficiencia de las alternativas utilizadas.

En la segunda parte del capítulo se propone la utilización de un algoritmo GNG al que se le han incorporado un grupo de modificaciones (GNGM). Estas modificaciones están encaminadas adaptar el algoritmo a su uso en el marco de la investigación. Las modificaciones tienen como objetivos: acelerar el aprendizaje en las primeras etapas, disminuir la importancia de la selección de algunos parámetros y evitar la proliferación de nodos en regiones densas. Para evitar la proliferación de nodos se incorporaron tres restricciones en la inserción, que manejando diferentes criterios, intentan lograr una homogeneidad en la estructura del modelo.

Capítulo 4

Grafos embebidos en espacios vectoriales

Los algoritmos de *clustering* han sido diseñados para ser aplicados principalmente sobre puntos $(x_1, \ldots, x_n) \in \mathbb{R}^n$. Esto se debe a que en muchos casos los datos son representados de esa manera. Sin embargo, en muchas otras ocasiones es necesario utilizar formas de representación más flexibles como los grafos. Esta forma de representación permite describir otras propiedades de los objetos, como las relaciones estructurales, etc.

El problema radica en cómo aplicar los algoritmos existentes a estos dominios, pues muchas de las operaciones que se utilizan en su funcionamiento se basan en las propiedades de los espacios vectoriales.

4.1 Grafos embebidos en espacios vectoriales mediante selección de prototipos

El enfoque de grafos embebidos fue propuesto originalmente en (Wilson et al., 2003). Este método está basado en la teoría algebraica de grafos y utiliza una matriz de descomposición espectral. En el enfoque propuesto en (Riesen et al., 2007) se hace un uso explícito de la distancia de grafos. Por esta razón, se pueden manejar varios tipos de grafos (etiquetados, sin etiquetar, dirigidos, no dirigidos, etc.) y se puede utilizar conocimiento específico del dominio en el que se trabaje, definiendo similitud entre

nodos y aristas mediante costos de edición. Así, se puede alcanzar un alto grado de robustez contra varios tipos de distorsiones de grafos.

La idea sobre la que se basa el método fue originalmente desarrollada para el problema de embeber un conjunto de vectores de rasgos en una matriz de similitud (Pekalska et al., 2006). En el enfoque propuesto en (Riesen et al., 2007) se introduce una extensión al dominio de los grafos.

Para el enfoque de grafos embebidos en espacios vectoriales se asume que se tiene un conjunto de entrenamiento de grafos etiquetados $\mathcal{G} = \{g_1, \ldots, g_n\}$ y una medida de similitud $d(g_i, g_j)$. Tras seleccionar un conjunto $\mathcal{P} = \{p_1, \ldots, p_m\}$ de $m < n$ prototipos de $\mathcal{G}$, se computa la similitud de un grafo $g \in \mathcal{G}$ a cada prototipo $p \in \mathcal{P}$. Esto conduce a m similitudes, $d_1 = d(g, p_1)$, $\ldots$, $d_m = d(g, p_m)$; las cuales pueden ser interpretadas como un vector m-dimensional $(d_1, \ldots, d_m)$. De esta manera se puede transformar cualquier grafo del conjunto de entrenamiento, así como cualquier grafo de los conjuntos de validación o de prueba, en un vector de números reales. Se puede señalar, que cualquier grafo del conjunto de entrenamiento que ha sido seleccionado como uno de los prototipos, es transformado en un vector $x = (x_1, \ldots, x_m)$ donde uno de sus componentes es cero.

Formalmente, si $\mathcal{G} = \{g_1, \ldots, g_n\}$ es un conjunto de grafos de entrenamiento y $\mathcal{P} = \{p_1, \ldots, p_m\} \subseteq \mathcal{G}$ es un conjunto de prototipos, el mapeado:

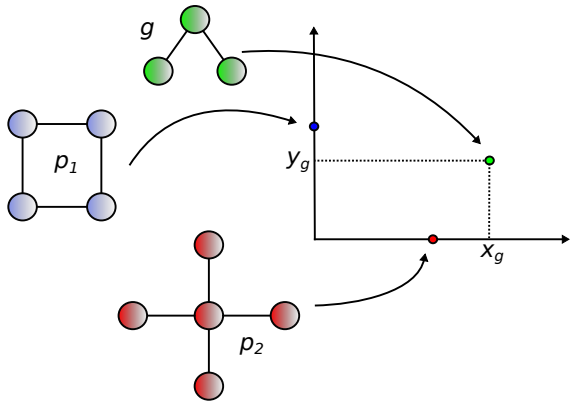
$$\varphi_m^{\mathcal{P}} : \mathcal{G} \to \mathbb{R}^m$$

se define como una función:

$$\varphi_m^{\mathcal{P}}(g) \to (d(g, p_1), \ldots, d(g, p_m)) \tag{4.1}$$

donde $d(g, p_i)$ es la distancia de similitud, en este caso la distancia de edición, entre el grafo g y el i-ésimo prototipo. En la figura 4.1 se muestra un ejemplo de mapeado de un grafo con respecto a dos prototipos.

El procedimiento descrito permite obtener un espacio vectorial donde cada eje se corresponde con un grafo prototipo $p_i \in \mathcal{P}$ y las coordenadas de un grafo embebido g son las distancias de g a los elementos de $\mathcal{P}$. Cualquier distancia de similitud puede ser empleada, lo cual posibilita que una cantidad significativa de tipos de grafos (dirigidos, no dirigidos,



$x_g = d(g, p_1),\ y_g = d(g, p_2)$

Figura 4.1 Grafos embebidos en un espacio bidimensional. Los grafos p_1 y p_2 son prototipos.

no etiquetados, etiquetados por nodos o aristas, etc...) sea aceptable para aplicar el procedimiento.

4.1.1 Selección de prototipos

En el contexto del procedimiento para embeber grafos en espacios vectoriales, la selección de los n prototipos $\mathcal{P} = \{p_1, \ldots, p_n\}$ es una etapa crítica, pues no solo los prototipos, sino también su número, afectan el resultado del mapeado $\varphi_m^{\mathcal{P}}(g)$. Una primera restricción en la utilización de prototipos viene dada por su cantidad, pues el empleo de muchos prototipos aumenta la dimensión de los vectores resultantes y, como consecuencia, afecta la eficiencia del procedimiento.

En la selección de prototipos también deben considerarse los efectos de la utilización de elementos similares o de *outliers*. Esto trae como resultado que se emplee información redundante o irrelevante en el mapeado de los grafos. Más formalmente, mientras más similares sean dos prototipos p_i y p_j de $\mathcal{P}$, más similares y, por tanto, redundantes, serán sus correspondientes entradas d_i y d_j en el mapeado de un grafo arbitrario $\varphi_m^{\mathcal{P}}(g)$. Por otro lado, asumiendo que un prototipo $p_i \in \mathcal{P}$ es un *outlier*, el valor $d(g, p_i)$ es grande para cualquier entrada $g \in \mathcal{G}$. Como consecuencia, para dos grafos arbitrarios g y g', las entradas correspon-

dientes $d(g, p_i)$ y $d(g', p_i)$ serán muy grandes y su diferencia relativa, insignificante y carente de poder discriminatorio.

En (Riesen, 2009) son analizadas un grupo de estrategias para la selección de prototipos. Todas las estrategias investigadas necesitan que el tamaño del conjunto de prototipos $\mathcal{P}$ sea un parámetro definido por el usuario. Así, el número de prototipos que se emplee debe ser determinado experimentalmente o elegido de manera arbitraria.

Con relación a la selección de prototipos como un procedimiento de selección de rasgos (en este caso los prototipos se corresponden con los rasgos), son aplicables las estrategias clásicas de evaluación de la calidad de los subconjuntos de rasgos: filtros y envolturas (*wrappers*). Los filtros evalúan cada rasgo de manera individual asignándole una puntuación de acuerdo a las características generales del conjunto de entrenamiento. Como regla, son seleccionados una cantidad fija de rasgos con la más alta puntuación, o alternativamente, los rasgos cuyas puntuaciones superen un umbral predefinido. En el enfoque de los *wrappers* la ejecución de un algoritmo de reconocimiento de patrones o un conjunto de validación independiente pueden ser empleados como criterios para la evaluación de los subconjuntos de rasgos.

En general, de los métodos estudiados en (Riesen, 2009), los que mejores resultados experimentales mostraron (en el caso de los aplicables a conjuntos de datos no etiquetados) fueron: BPS (*Border Prototype Selector*) y SPS (*Spanning Prototype Selector*). BPS selecciona como prototipos elementos ubicados en el borde del conjunto de entrenamiento. El conjunto de prototipos es construido iterativamente a partir de la selección de aquellos elementos que maximizan la distancia al resto de los elementos.

El método de selección de prototipos seleccionado en este trabajo es el SPS (ver algoritmo 4.1). La concepción del algoritmo SPS es mucho más consecuente con la idea de la selección de prototipos distribuidos uniformemente en el conjunto de entrenamiento, para lograr captar la mayor parte de la diversidad del conjunto de los datos.

El primer paso del algoritmo SPS es la selección del elemento g_{mdn} cuya suma de las distancias al resto del conjunto de grafos $\mathcal{G}$ es mínima, o sea:

Algoritmo 4.1 Selección de prototipos mediante SPS.

1. Encontrar el grafo g_{mdn}.
2. Incluir a g_{mdn} en el conjunto de prototipos $\mathcal{P}$.
3. Encontrar el elemento p' que más lejano se encuentre de los prototipos $\mathcal{P}$, e incluirlo en $\mathcal{P}$.
4. Si no se ha alcanzado la cantidad de elementos definida para $\mathcal{P}$, volver al paso 3.

$$g_{mdn} = \arg \min_{g_1 \in \mathcal{G}} \sum_{g_2 \in \mathcal{G}} d(g_1, g_2) \tag{4.2}$$

lo que intuitivamente ubica a este elemento en el centro del conjunto de datos.

Una vez encontrado el grafo g_{mdn}, se realiza un proceso iterativo de selección del elemento p' que más lejano se encuentre de los prototipos anteriormente seleccionados, o sea:

$$p' = \arg \max_{g \in \mathcal{G} \setminus \mathcal{P}} \min_{p \in \mathcal{P}} d(g, p) \tag{4.3}$$

Este proceso se repite hasta alcanzar la cantidad de prototipos definida a priori.

4.1.2 Clasificación de grafos embebidos en espacios vectoriales

El objetivo de la aplicación del procedimiento de grafos embebidos en espacios vectoriales es la proyección de un conjunto de grafos en un espacio vectorial. Una vez lograda esta representación vectorial, se pueden emplear las distintas herramientas de *clustering* disponibles en estos entornos, que en el marco de esta investigación es el algoritmo GNGM, propuesto en el capítulo 3.

Como resultado de la aplicación del algoritmo GNGM se va a obtener un modelo mediante el cual se pueden hacer agrupaciones de los datos en el espacio vectorial, que se van a corresponder con las agrupaciones en el conjunto de grafos. Esto es, los grafos cuyas proyecciones en el espacio

vectorial estén agrupadas en un mismo *cluster*, van a estar a su vez agrupados en un mismo *cluster* (veáse figura 4.2). Además, también se van a corresponder las relaciones de vecindad presentes en el modelo, o sea, los *clusters* vecinos en el espacio vectorial, van a ser *clusters* vecinos en el espacio de los grafos.

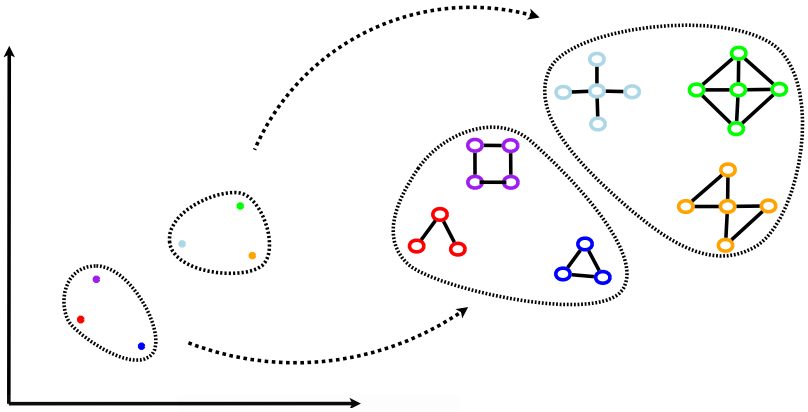


Figura 4.2 Correspondencia entre el agrupamiento en el espacio vectorial (izquierda) y el conjunto de los grafos (derecha).

La calidad de las agrupaciones, en general, no va a ser la misma cuando se evalúen en el espacio vectorial, que cuando se evalúen en el espacio de los grafos. Si se aplica una medida que evalúe la agrupación de elementos similares en un mismo *cluster* y elementos distintos en *clusters* diferentes, esta medida empleará la distancia de grafos en un caso y la distancia euclidiana en el otro. De manera general, la calidad del agrupamiento será peor en el caso del espacio de los grafos, pues en el proceso de embeber estos en el espacio vectorial, casi siempre ocurrirá una pérdida de información que afectará las relaciones de distancias relativas entre los dos espacios, o sea, elementos que están cercanos en el espacio de los grafos pueden estar relativamente más lejanos en el espacio vectorial, o viceversa.

A partir de lo anterior, se puede deducir que mientras mejor es la calidad del procedimiento de embebido de grafos en un espacio vectorial, más próxima será la calidad del agrupamiento en el espacio de los grafos a la calidad del agrupamiento en el espacio vectorial. La calidad del

procedimiento de embebido va a depender, a su vez, de la calidad de la medida de distancia entre grafos que se emplee y de la calidad del proceso de selección de prototipos.

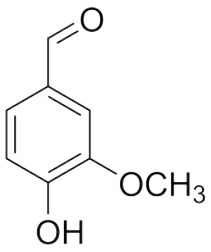
4.2 Experimentación

4.2.1 Conjunto de experimentación

Para comprobar la efectividad del proceso de grafos embebidos en espacios vectoriales, y posteriormente, poner a prueba la propuesta general de solución que se propone en esta investigación fue conformado un conjunto de experimentación compuesto por 11 347 moléculas. Este conjunto, a su vez, ha sido construido a partir de la agrupación de conjuntos de datos disponibles públicamente con fines comerciales o científicos (Stahl and Rarey, 2001b; Delaney, 2004; Huuskonen, 2000; Kazius et al., 2005; Cavalli et al., 2002; Feng et al., 2003). Estos conjuntos han sido utilizados en la validación de diversas publicaciones científicas en temas vinculados al diseño de medicamentos (Stahl and Rarey, 2001a; Rarey and Stahl, 2001; Liao et al., 2007; Wang et al., 2009; Baurin et al., 2004; Wang et al., 2007; Verdonk et al., 2004; Riesen, 2009). La selección de este conjunto de moléculas está vinculada a la motivación inicial de esta investigación.

La información de las moléculas se encuentra almacenada en ficheros en los formatos SMILES (Weininger, 1990) o SDF (Dalby et al., 1992). Ambos formatos son ampliamente empleados para la manipulación computacional de moléculas, pues están diseñados para captar en mayor o menor medida la información fundamental que define un compuesto químico (átomos y enlaces). Cualquiera de los dos formatos permiten, entonces, conformar un grafo con esta información. En la figura 4.3 se muestra la representación de una molécula en cada uno de los formatos. El formato SMILES es un formato bastante simplificado, mientras que SDF incluye información más detallada de los átomos y enlaces (cargas, posición espacial, etc.).

Estructura de la vanilina:



SMILES:

```
O=Cc1ccc(O)c(OC)c1
```

SDF:

```
 11 11  0  0  0  0  0  0  0  0999 V2000
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  2  0
  2  3  1  0
  3  4  2  0
  4  5  1  0
  5  6  2  0
  6  7  1  0
  6  8  1  0
  9 10  1  0
  8  9  1  0
  8 11  2  0
 11  3  1  0
M  END
```

Figura 4.3 Formatos de almacenamiento digital de una molécula.

4.2.2 Medida de similitud

El método de grafos embebidos en espacios vectoriales requiere la definición de una medida de similitud entre grafos. En el contexto de esta investigación se ha empleado la distancia de Tanimoto entre las representaciones en *fingerprints* de las moléculas.

En muchos contextos son utilizados como representación de grafos los *fingerprints* (Flower, 1998; Eckert and Bajorath, 2006), pues constituyen una forma compacta de los mismos, y sobre la cual se pueden realizar un grupo de operaciones de manera eficiente. Existen varias medidas de similitud que pueden ser empleadas entre elementos representados como *fingerprints* (Dice, Cosine, Sokal, Russel, Kulczynski y McConnaughey) pero la más utilizada es la distancia de Tanimoto (Willett et al., 1998; Cramer et al., 2004).

La mayoría de estas medidas de similitud se obtienen a partir de las respectivas definiciones de coeficientes de similaridad. La distancia de Tanimoto parte del coeficiente de Tanimoto, que para el caso de vectores binarios como los *fingerprints* se define como:

$$f_t(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{4.4}$$

donde A y B son vectores binarios. Entonces, la distancia de Tanimoto se define como:

$$d_t(A, B) = 1 - f_t(A, B) \tag{4.5}$$

Una de las razones por las cuales la distancia de Tanimoto está tan extendida es porque al cumplirse la desigualdad triangular, esta distancia constituye una métrica (Lipkus, 1999).

En la figura 4.4 se muestra el resultado de emplear el método de grafos embebidos en espacios vectoriales con el conjunto de experimentación. En este caso, el conjunto de prototipos está formado por tres elementos y se ha empleado la distancia de Tanimoto como medida de similitud entre las moléculas.

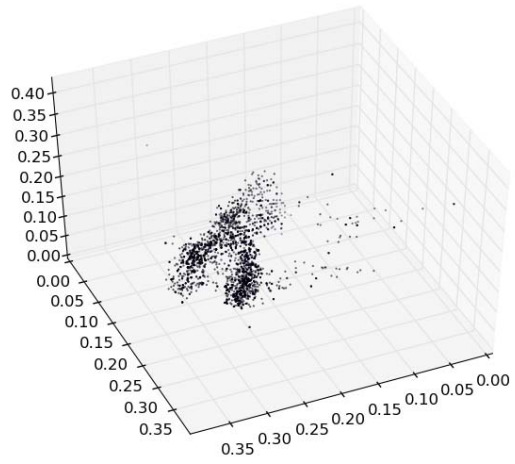


Figura 4.4 Proyección de los grafos correspondientes a las moléculas en un espacio vectorial de tres dimensiones. Cada dimensión se corresponde con la distancia a un prototipo.

4.2.3 Resultados experimentales

En este epígrafe se exponen los experimentos realizados con el objetivo de comprobar la eficacia del procedimiento de embeber grafos en espacios vectoriales con el conjunto de datos seleccionado.

Evaluación del método de selección de prototipos SPS

Este grupo de experimentos está encaminado a evaluar de manera general el método de grafos embebidos en espacios vectoriales, haciendo énfasis en el estudio del método de selección de prototipos SPS.

En estos experimentos se utiliza una medida que evalúa el grado de preservación de las distancias relativas entre elementos en el conjunto de grafos y sus representaciones en el espacio vectorial, cuando se aplica el método de grafos embebidos. El empleo de esta medida se justifica en el hecho de que una parte importante de los algoritmos de agrupamiento está vinculada con la selección de los elementos más próximos a un punto

determinado. Así, en la medida que mejor se preserven las distancias relativas, más similares podrán ser los resultados del agrupamiento en el espacio original y en el espacio vectorial.

La determinación de la medida de preservación de las distancias relativas se define como: dado tres grafos aleatorios g_1, g_2 y g_3; se calculan sus respectivas proyecciones en el espacio vectorial $\varphi(g_1)$, $\varphi(g_2)$ y $\varphi(g_3)$; y se comprueba si se cumple:

$$\frac{d_t(g_1, g_2) - d_t(g_1, g_3)}{\|\varphi(g_1) - \varphi(g_2)\| - \|\varphi(g_1) - \varphi(g_3)\|} > 0 \tag{4.6}$$

donde $d_t(g_i, g_j)$ es la distancia entre los grafos g_i y g_j según la medida de similitud definida. O sea, si se toma un grafo g_1 como referencia, se determina cual de los grafos g_2 o g_3 está más cerca, y se comprueba si esa misma relación se cumple entre las respectivas proyecciones $\varphi(g_1)$, $\varphi(g_2)$ y $\varphi(g_3)$.

La operación de generar tres grafos aleatorios y efectuar la comprobación anterior se realiza un cantidad elevada de veces para lograr una estimación precisa del grado de preservación de las distancias relativas.

Los experimentos realizados están diseñados específicamente para estudiar la influencia de la cantidad de prototipos que se seleccionan, la sensibilidad y la importancia del método SPS en la calidad del proceso de grafos embebidos.

El primer procedimiento seguido en estos experimentos consiste en ejecutar el algoritmo SPS y aplicar el método de grafos embebidos para una cantidad creciente de prototipos. Los pasos del procedimiento son:

1. Seleccionar el primer prototipo según el algoritmo SPS y añadirlo al conjunto de prototipos $\mathcal{P}$.

2. Aplicar el método de grafos embebidos con los prototipos $\mathcal{P}$ y el conjunto $\mathcal{G}$.

3. Seleccionar el próximo prototipo del conjunto $\mathcal{G} \setminus \mathcal{P}$ según el algoritmo SPS y añadirlo al conjunto de prototipos $\mathcal{P}$.

4. Volver al paso 2.

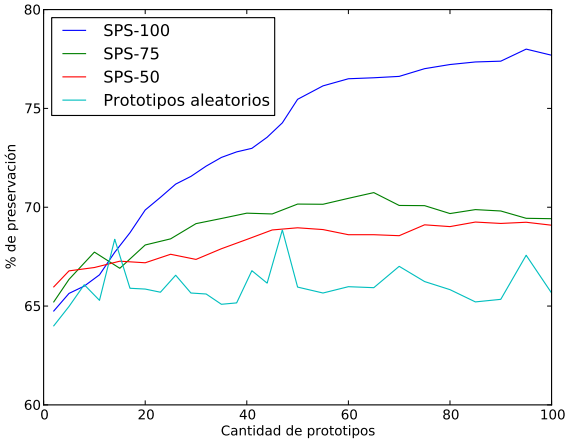
El segundo y tercer experimentos fueron realizados modificando el conjunto de donde se seleccionan los prototipos (paso 3.), sustituyendo el conjunto $\mathcal{G}$ por los conjuntos $\mathcal{G}_{75}$ y $\mathcal{G}_{50}$, conformados con muestras aleatorias correspondientes al 75 % y al 50 % de los datos, respectivamente:

1. Seleccionar el primer prototipo según el algoritmo SPS y añadirlo al conjunto de prototipos $\mathcal{P}$.
2. Aplicar el método de grafos embebidos con los prototipos $\mathcal{P}$ y el conjunto $\mathcal{G}$.
3. Seleccionar el próximo prototipo del conjunto $\mathcal{G}_{75} \setminus \mathcal{P}$ ($\mathcal{G}_{50} \setminus \mathcal{P}$) según el algoritmo SPS y añadirlo al conjunto de prototipos $\mathcal{P}$.
4. Volver al paso 2.

Por último se efectuó un experimento eliminando la selección de prototipos por el método SPS (pasos 1. y 3.), sustituyéndolo por una selección aleatoria:

1. Seleccionar aleatoriamente un elemento de $\mathcal{G}$ y añadirlo al conjunto de prototipos $\mathcal{P}$.
2. Aplicar el método de grafos embebidos con los prototipos $\mathcal{P}$ y el conjunto $\mathcal{G}$.
3. Seleccionar aleatoriamente un elemento de $\mathcal{G} \setminus \mathcal{P}$ y añadirlo al conjunto de prototipos $\mathcal{P}$.
4. Volver al paso 2.

Los resultados de este conjunto de experimentos se observan en la figura 4.5. El análisis del primer experimento (SPS-100) permite apreciar cómo en la medida que se utilizan más prototipos generados por el algoritmo SPS, aumenta el grado de preservación de las distancias relativas. Este crecimiento es considerable hasta que se incorporan alrededor de 50 prototipos, y a partir de este punto el crecimiento es menos significativo.



SPS-100: 100 %, SPS-75: 75 %, SPS-50: 50 %

Figura 4.5 Comparación de la preservación de las distancias relativas entre objetos en el conjunto de grafos y sus proyecciones en el espacio vectorial en el algoritmo SPS, seleccionando prototipos de distintas muestras de los datos.

Los resultados del segundo y tercer experimentos se muestran en la figura 4.5 con las curvas SPS-75 Y SPS-50 respectivamente. La selección de prototipos empleando subconjuntos aleatorios de los datos tiene el objetivo de que el método no seleccione los mejores prototipos, y mientras menor sea el subconjunto de selección, menos probabilidad hay de elegir los prototipos óptimos. Esta idea se ve reflejada en los resultados, pues la curva correspondiente a la selección en el conjunto $\mathcal{G}_{75}$ se mantiene por encima de la curva $\mathcal{G}_{50}$, y ambas bastante por debajo de SPS-100.

En la figura 4.5 también se muestran los resultados del experimento que emplea una selección aleatoria de prototipos. Esta curva posee un comportamiento irregular, y el porcentaje de preservación de las distancias relativas refleja una pérdida de información mayor, con respecto a las variantes que seleccionan los prototipos mediante el algoritmo SPS.

El resultado global de estos experimentos refleja la efectividad del método SPS como forma de selección de prototipos con respecto a la

preservación de las distancias relativas. Los resultados mostraron que para el conjunto de moléculas empleado, la preservación de estas distancias alcanza entre un 75 % y un 78 % cuando se selecciona una cantidad de prototipos superior a los 48 elementos.

Un elemento a señalar es el hecho de que a partir de los 48 prototipos, la mejora en la preservación de las distancias relativas no es significativa. Este aspecto es importante pues el aumento en la cantidad de prototipos conduce a un aumento en los tiempos de ejecución, sin que esto implique una mejora considerable en la calidad del proceso de embebido de los grafos en el espacio vectorial.

Aplicación del algoritmo GNGM sobre conjuntos de grafos embebidos en espacios vectoriales

El siguiente experimento está orientado a comprobar la efectividad de la aplicación del algoritmo GNGM, sobre elementos en un espacio vectorial obtenido a partir del procedimiento de embebido de grafos.

En este grupo de experimentos se utiliza el índice C (*C index*) (Hubert and Schultz, 1976) como medida de evaluación de la calidad del proceso de *clustering*. Esta medida es apropiada en este contexto pues permitirá la evaluación de los *clusters* conformados, tanto en el espacio vectorial como en el espacio de los grafos.

Para computar el índice C se define la función $c(\cdot)$ de la siguiente manera:

$$c(x_i, x_j) = \begin{cases} 1 & \text{si } x_i \text{ y } x_j \text{ pertenecen al mismo } \textit{cluster} \\ 0 & \text{en otro caso} \end{cases} \tag{4.7}$$

Además, se define Γ como la suma de todas las distancias de los objetos pertenecientes al mismo *cluster*, o sea:

$$\Gamma = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(x_i, x_j) c(x_i, x_j) \tag{4.8}$$

donde $d(\cdot)$ es la función de distancia entre dos objetos.

Se define también a como la cantidad de pares de objetos que están en un mismo *cluster*, es decir:

$$a = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c(x_i, x_j) \tag{4.9}$$

Finalmente, se define el índice C como:

$$C = \frac{\Gamma - min}{max - min} \tag{4.10}$$

donde min es la suma de las a menores y max la suma de las a mayores distancias $d(x_i, x_j)$ donde $x_i \neq x_j$.

El numerador del índice C cuantifica cuántos pares de objetos de los a pares más cercanos pertenecen al mismo *cluster*. El denominador se utiliza para asegurar que $0 \leq C \leq 1$. Mientras menor sea el valor del índice C, más frecuentemente los pares con distancias pequeñas entre ellos pertenecen al mismo *cluster*, o sea, mayor es la calidad del agrupamiento.

El experimento que se presenta en este epígrafe tiene como objetivo evaluar la calidad, en términos del índice C, de la agrupación de los elementos del conjunto de experimentación. Para ello se usa la combinación del procedimiento de grafos embebidos en espacios vectoriales y el algoritmo de *clustering* propuesto para agrupar las proyecciones de los grafos en el espacio vectorial. Además, se estudia la influencia de la cantidad de prototipos en la calidad del agrupamiento.

El procedimiento experimental consiste en:

1. Seleccionar el primer prototipo según el algoritmo SPS y añadirlo al conjunto de prototipos $\mathcal{P}$.

2. Aplicar el método de grafos embebidos con los prototipos $\mathcal{P}$ y el conjunto $\mathcal{G}$.

3. Aplicar el algoritmo GNGM.

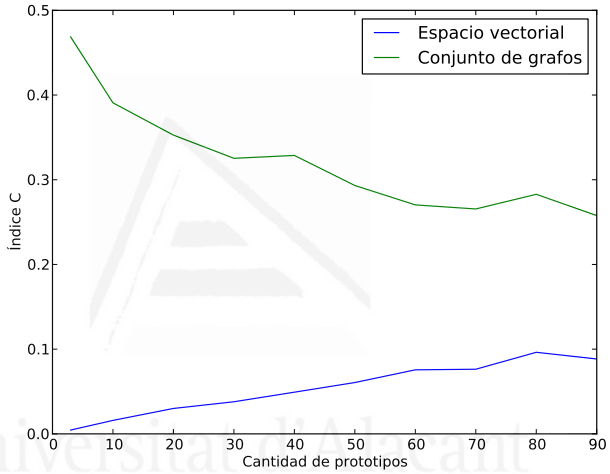
4. Agrupar los elementos del conjunto de grafos según la agrupación de sus proyecciones en el espacio vectorial.

5. Calcular el índice C para el agrupamiento en el espacio vectorial y en el conjunto de grafos.

6. Seleccionar el próximo prototipo del conjunto $\mathcal{G} \setminus \mathcal{P}$ según el algoritmo SPS y añadirlo al conjunto de prototipos $\mathcal{P}$.

7. Volver al paso 2.

Los resultados del experimento se observan en la figura 4.6. Aquí se puede comprobar cómo el valor del índice C para el agrupamiento de las proyecciones en el espacio vectorial aumenta con el número de prototipos que se emplean en el procedimiento de embebido. Este resultado se puede explicar por el aumento de las dimensiones del espacio vectorial resultante y, por tanto, de la complejidad de los datos a agrupar. El índice C parte de valores muy pequeños hasta alcanzar valores cercanos a 0.1, lo cual indica buenos resultados en todos los casos.



Parámetros GNGM ($\lambda = 50$, $ed_{max} = 50$, $\alpha = 0.5$, $\alpha' = 0.9$, $\varphi = 0.05$, $\gamma = 0.05$, $\sigma = 0.05$)

Figura 4.6 Relación entre el valor del índice C y la cantidad de prototipos para el agrupamiento en el espacio vectorial y el conjunto de grafos.

En el caso del valor del índice C para el agrupamiento en el conjunto de los grafos, este disminuye con el aumento de la cantidad de prototipos que se utilizan en el procedimiento de embebido. Los valores iniciales son relativamente altos, pero al aumentar el número de prototipos que

se utilizan y, por tanto, captar mayor cantidad de información que refleje la diversidad del conjunto de los grafos, los valores disminuyen en casi la mitad.

De manera análoga a lo que se apreciaba en el experimento sobre la preservación de las distancias relativas, en este experimento se observa como a partir de cierto punto la disminución del valor del índice C es poco significativa con respecto al aumento de la cantidad de prototipos empleados en el procedimiento de grafos embebidos. A partir del resultado alcanzado se seleccionó el conjunto de datos vectoriales obtenidos utilizando 60 prototipos, así como el modelo obtenido a partir de la aplicación del algoritmo GNGM, para las pruebas realizadas en la etapa de búsqueda de la solución general que se propone en la investigación.

4.3 Implementación en paralelo de la selección de prototipos

La etapa computacionalmente más costosa dentro del método de grafos embebidos en espacios vectoriales es el proceso de selección de prototipos mediante el algoritmo SPS. Una vez que el conjunto de prototipos está definido, la proyección de cada elemento en el espacio vectorial es un proceso sencillo de distribuir.

El algoritmo SPS se divide, a su vez, en dos etapas: la selección de un grafo centralmente ubicado, que será el primer prototipo, y la selección progresiva de prototipos, buscando el grafo que más lejano se encuentre del conjunto de prototipos.

En la figura 4.7 se muestra el esquema del procedimiento que se emplea para paralelizar el algoritmo SPS. En primer lugar, se distribuye el cálculo de las distancias entre cada par de elementos, así como las sumas de las distancias de cada elemento al resto (ver ecuación 4.2). Así se determina el primer prototipo. Seguidamente, se distribuye el cálculo del elemento cuya mínima distancia al conjunto de prototipos es máxima (ver ecuación 4.3), asociado a la selección progresiva de prototipos.

En la figura 4.8 se muestra la distribución de las tareas de cómputo asociadas a la selección del primer prototipo entre las unidades de procesamiento. Los cálculos de distancia entre cada par de elementos están

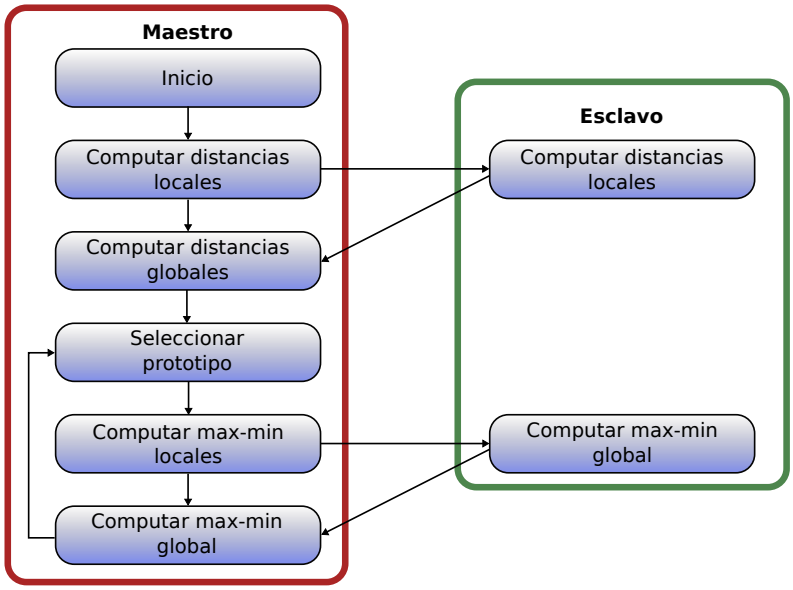


Figura 4.7 Diagrama del proceso de paralelización de la selección de prototipos mediante el algoritmo SPS.

ubicados en el triángulo superior de la matriz. La elección del procesador en el que se realiza el cálculo de la distancia entre dos elementos (i, j, $i > j$) del conjunto de datos se describe mediante la función $k(i,j)$:

$$k(i,j) = \begin{cases} i \cdot \left\lfloor \frac{N}{M} \right\rfloor & \text{si } i \,\%\, \left\lfloor \frac{M}{N} \right\rfloor < j \,\%\, \left\lfloor \frac{M}{N} \right\rfloor \\ j \cdot \left\lfloor \frac{N}{M} \right\rfloor & \text{si } i \,\%\, \left\lfloor \frac{M}{N} \right\rfloor \geq j \,\%\, \left\lfloor \frac{M}{N} \right\rfloor \end{cases} \tag{4.11}$$

donde M es la cantidad de datos y N es la cantidad de procesadores ($M \gg N$).

Además del cálculo de las distancias entre pares de elementos, es necesario determinar la suma de las distancias de un elemento al resto. En cada procesador se realizan las sumas parciales para cada elemento para el que se ha calculado una distancia. Así, al terminar este cómputo, estas sumas parciales son enviadas al nodo maestro, donde se realiza el cómputo global y la determinación del elemento cuya suma es mínima, que es seleccionado como primer prototipo.

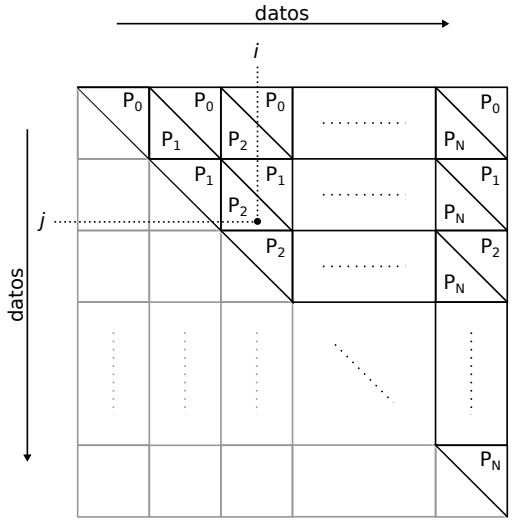


Figura 4.8 Distribución del cómputo asociado a la selección del grafo más centralmente ubicado entre las unidades de procesamiento $(P_0, \cdots, P_n)$.

El esquema empleado garantiza que se distribuye de manera equitativa la cantidad de cálculos de distancia y sumas parciales en cada procesador, además de que cada procesador puede determinar de manera independiente la porción de los datos que le corresponde procesar.

El segundo grupo de tareas que se distribuye en paralelo son las asociadas a la selección de un prototipo a partir de la búsqueda del elemento que más lejano se encuentra del conjunto de prototipos. En cada unidad de procesamiento k se va a computar el candidato a prototipo:

$$p'_k = \arg \max_{g \in \mathcal{G}_k \setminus \mathcal{P}} \min_{p \in \mathcal{P}} d(g, p) \tag{4.12}$$

donde $\mathcal{G}_k$ es el subconjunto de elementos que se analizan en la unidad de procesamiento k:

$$\mathcal{G}_k = \left\{ g_i \in \mathcal{G}, \quad k \cdot \left\lfloor \frac{M}{N} \right\rfloor \leq i < (k+1) \cdot \left\lfloor \frac{M}{N} \right\rfloor \right\} \tag{4.13}$$

Las unidades de procesamiento, tras haber encontrado sus máximos-mínimos locales, envían esta información al nodo maestro, donde se determina el máximo global, y se selecciona el próximo prototipo. El nuevo

prototipo es enviado a cada unidad de procesamiento para actualizar el conjunto de prototipos que se utilizará en la siguiente iteración.

4.3.1 Resultados experimentales

En este epígrafe se presentan los experimentos realizados con el objetivo de comprobar las mejoras en eficiencia que se logran con la implementación en paralelo del algoritmo SPS. Se efectuaron experimentos independientes para las dos etapas fundamentales del algoritmo: la selección del elemento ubicado en el centro del conjunto y la selección del resto de los prototipos. En cada cada caso se ejecuta el algoritmo para una cantidad creciente de elementos, utilizando el conjunto de datos de moléculas descrito en el epígrafe 4.2.1.

El primer experimento compara la diferencia de tiempo de ejecución entre las implementaciones secuencial y en paralelo de la etapa de búsqueda del primer prototipo. Los resultados se observan en la figura 4.9. Aquí se aprecia cómo el tiempo de ejecución mejora notablemente con el empleo de la implementación en paralelo.

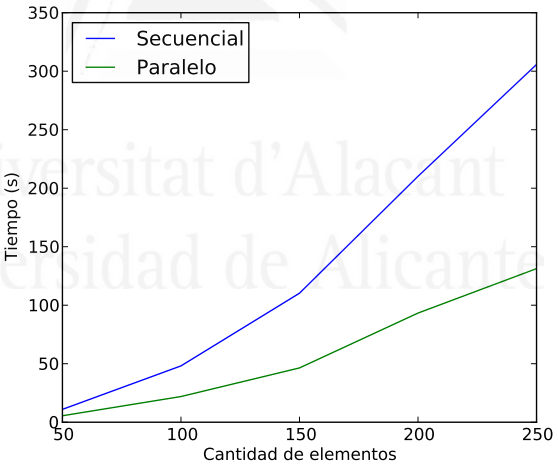
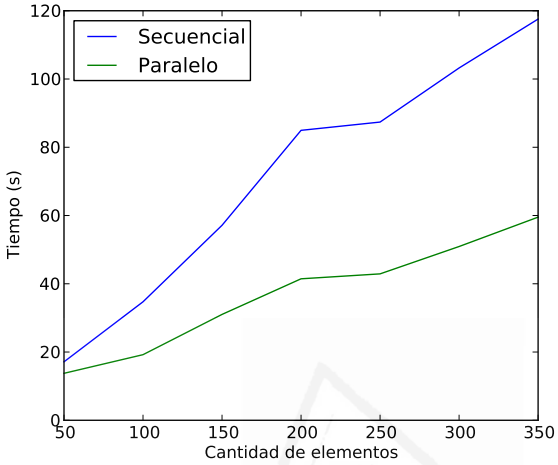


Figura 4.9 Comparación del tiempo de ejecución de la selección del primer prototipo en el algoritmo SPS entre las implementaciones secuencial y en paralelo.

En el segundo experimento se compara la diferencia de tiempo de ejecución entre las implementaciones secuencial y en paralelo de la selección iterativa de prototipos, para la selección de los 10 primeros prototipos. Los resultados se observan en la figura 4.10, y de manera análoga al experimento anterior, la implementación en paralelo reduce considerablemente los tiempos de ejecución.



En cada ejecución se seleccionan 10 prototipos.

Figura 4.10 Comparación del tiempo de ejecución de la selección iterativa de prototipos en el algoritmo SPS entre las implementaciones secuencial y en paralelo.

Los resultados obtenidos en estos experimentos confirman la ventaja que representa el uso de la implementación en paralelo descrita para el algoritmo SPS.

4.4 Sumario

En este capítulo se describe el método de grafos embebidos en espacios vectoriales. Este método es parte de la solución que se propone en esta investigación pues permite la aplicación del algoritmo de *clustering* propuesto sobre un conjunto de grafos.

En el capítulo se presenta un conjunto de experimentación formado

por moléculas, que es utilizado para comprobar la viabilidad del empleo del método en el contexto de esta investigación.

Los resultados experimentales que se exponen permiten concluir que es factible la aplicación de la metodología consistente en aplicar el método de grafos embebidos en espacios vectoriales a un conjunto de moléculas, para la posterior aplicación del algoritmo de *clustering* propuesto sobre las proyecciones de las moléculas en el espacio vectorial. El objetivo final es obtener un agrupamiento con preservación de la topología sobre el conjunto de moléculas.

A partir del análisis de los experimentos, se seleccionó el espacio vectorial obtenido a partir de la aplicación del método de grafos embebidos en el conjunto de datos utilizando 60 prototipos, y el modelo resultante de la aplicación del algoritmo GNGM sobre este espacio vectorial, como base de la experimentación de la etapa de búsqueda de la propuesta de solución general.

Finalmente, se describe la implementación en paralelo utilizada en el proceso de selección de prototipos SPS, y se exponen los resultados que confirman su eficiencia.

Capítulo 5

Organización del proceso de búsqueda

En este capítulo se aborda el proceso de búsqueda de un elemento óptimo respecto a una determinada función objetivo, dentro de los elementos pertenecientes a un conjunto de datos previamente organizado según los procedimientos expuestos en los capítulos anteriores. Esta búsqueda se realiza dentro del modelo generado a partir de la aplicación del algoritmo GNGM, y se orienta a partir de la estimación del valor de la función objetivo en los nodos del modelo mediante la evaluación progresiva de sus elementos.

5.1 Búsqueda basada en la estimación de la calidad de los nodos

En los capítulos anteriores se abordó cómo organizar un grupo de datos pocos estructurados, en este caso grafos, mediante el empleo combinado del método de grafos embebidos en espacios vectoriales y el algoritmo GNGM. El resultado es que los elementos del conjunto de datos están clasificados según el modelo que se obtiene del algoritmo GNGM. A partir de esta clasificación, se puede presumir que los elementos que se encuentran en un mismo nodo son similares; y a su vez, la similitud entre elementos ubicados en distintos nodos dependerá de la cercanía

entre dichos nodos.

El siguiente paso dentro de la propuesta de solución general es encontrar un elemento del conjunto de datos que sea óptimo con respecto a una función objetivo. Esta función objetivo debe tener como característica que elementos similares presenten valores similares de la función. Además, un aspecto distintivo de las funciones que se consideran en este trabajo es que presentan una evaluación computacionalmente costosa, y por tanto, el tiempo que tarda la evaluación de todos los elementos es mucho mayor que el tiempo que se considera razonable para obtener una solución (ver ecuación 1.1).

En este trabajo se proponen métodos que se basan en la idea de guiar la búsqueda según una estimación de la calidad de los nodos del modelo resultante de la organización de los datos. La calidad de un nodo se estima por el promedio los valores de la función objetivo de los elementos del nodo.

5.1.1 Muestreo y búsqueda

El método de búsqueda general que se propone utiliza ideas provenientes de los algoritmos Metropolis-Hastings y *simulated annealing*.

Metropolis-Hastings

El algoritmo Metropolis-Hastings (MacKay, 2003) es un método de cadena de Markov de Monte-Carlo (MCMC) (Gilks et al., 1996) para obtener una secuencia de muestras aleatorias para las cuales es difícil realizar un muestreo de manera directa (algoritmo 5.1).

Suponiendo que el objetivo es obtener muestras aleatorias para una distribución $P(x)$, el algoritmo Metropolis-Hastings genera perturbaciones aleatorias del estado actual, y las acepta o rechaza atendiendo a cuánto se afecta la probabilidad del estado.

El algoritmo Metropolis-Hastings se define usando dos familias de distribuciones auxiliares: Q y R. $Q = (q_{ij})$ es la distribución de selección; donde q_{ij} es la probabilidad de seleccionar el estado s_j desde el estado actual s_i. $R = (r_{ij})$ es la distribución de aceptación; donde r_{ij} es la probabilidad de aceptar el estado s_j estando en el estado s_i, habiendo

seleccionado a s_j como posible próximo estado. Se debe cumplir que $q_{ij} \geq 0$, $r_{ij} \geq 0$ y $\sum_j q_{ij} = 1$. En la mayor parte de las aplicaciones prácticas, se puede asumir que Q es simétrica ($q_{ij} = q_{ji}$), pero esta hipótesis se puede relajar.

Algoritmo 5.1 Metropolis-Hastings.

1. Seleccionar un estado inicial s_i.
2. Seleccionar un estado aleatorio s_j de acuerdo con la distribución q_{ij}.
3. Aceptar s_j con probabilidad r_{ij}. Esto es, $s_i^{t+1} = s_j^t$ con probabilidad r_{ij} y $s_i^{t+1} = s_i^t$ con probabilidad $1 - r_{ij}$.
4. Repetir los pasos del 2 al 3 tantas veces como se necesite.

La distribución de aceptación en el algoritmo Metropolis-Hastings se define como:

$$r_{ij} = \text{mín}\left(1, \frac{P(s_j)}{P(s_i)}\right) \tag{5.1}$$

Cuando $P(s)$ se expresa en términos de una función de energía, la distribución de aceptación se puede reescribir como:

$$r_{ij} = \text{mín}\left(1, e^{-\frac{E(s_j)-E(s_i)}{kT}}\right) \tag{5.2}$$

donde $E(s)$ es la energía en el estado s, k la constante de Boltzmann y T es un parámetro llamado "temperatura". Como resultado de esta transformación se obtiene la versión más común del algoritmo, que se muestra en el algoritmo 5.2.

Simulated annealing

Simulated annealing (SA) (Kirkpatrick et al., 1983) es un algoritmo de optimización multiobjetivo de propósito general inspirado en la mecánica estadística (algoritmo 5.3). Este combina las ideas de MCMC, como el algoritmo Metropolis-Hastings, con un esquema de enfriamiento de la temperatura. El nombre de este método tiene sus orígenes en la metalurgia, donde los metales que son templados (enfriamiento lento) exhiben

Algoritmo 5.2 Metropolis-Hastings, versión más común.

1. Seleccionar un estado inicial s_i.
2. Seleccionar un estado aleatorio s_j de acuerdo con la distribución q_{ij}.
3. Si $\text{mín}(1, e^{-\frac{E(s_j)-E(s_i)}{kT}}) > random(0,1)$ entonces $s_i^{t+1} = s_j^t$; si no $s_i^{t+1} = s_i^t$.
4. Repetir los pasos del 2 al 3 tantas veces como se necesite.

propiedades de dureza superiores a los metales enfriados rápidamente. La mayor dureza macroscópica está asociada con estados moleculares internos de baja energía.

Algoritmo 5.3 *Simulated Annealing.*

1. Seleccionar una solución inicial x_i.
2. Generar una nueva solución a partir de una función de vecindad $x_j = vecindad(x_i)$.
3. Actualizar el óptimo global.
4. Si $\text{mín}(1, e^{-\frac{E(x_j)-E(x_i)}{kT}}) > random(0,1)$ entonces aceptar la solución $x_i^{t+1} = x_j^t$; si no permanecer en la solución actual $x_i^{t+1} = x_i^t$.
5. Decrementar T según el esquema de enfriamiento.
6. Repetir los pasos del 2 al 5 hasta que se alcance una condición de parada.

En problemas complejos de optimización combinatoria es usual quedar atrapados en óptimos locales. El objetivo de SA es que el algoritmo tenga más opciones de explorar el espacio de búsqueda, lo que se logra aceptando movimientos que producen soluciones de menos calidad, con una probabilidad que depende del parámetro "temperatura". Cuando la temperatura es alta, el algoritmo se comporta como una búsqueda aleatoria, o sea, acepta cualquier transición entre estados, sin importar si es buena o mala, lo que permite la exploración del espacio de

búsqueda. Mediante el empleo de un mecanismo de enfriamiento se disminuye gradualmente la temperatura. El algoritmo se comporta como una búsqueda *hill climbing* (Duda et al., 2001) cuando la temperatura tiene valores cercanos a 0, o sea, intensificando la búsqueda de una mejor solución. Si este proceso se realiza con el tiempo suficiente existe una alta probabilidad de alcanzar una solución óptima globlal (Ansari and Hou, 1997).

El algoritmo escapa de las soluciones óptimas locales moviéndose con cierta probabilidad a aquellas soluciones que son de menos calidad que la actual, y de esta manera, se crean oportunidades de una exploración más amplia del espacio de búsqueda. La probabilidad de aceptar una solución de menos calidad, $P(T)$, se comporta de acuerdo a la distribución de Boltzmann:

$$P(T) = e^{-\frac{E(s_j)-E(s_i)}{kT}} \tag{5.3}$$

tal y como se utiliza para el movimiento entre estados en el algoritmo Metropolis-Hastings. En muchas implementaciones, k se toma como un factor de escala para mantener la temperatura entre 0 y 1, si se desea que T se mantenga en ese intervalo.

En las primeras etapas del algoritmo se necesita encontrar un valor de T razonable, de manera que la mayoría de las transiciones sean aceptadas. Existen diversos tipos de esquemas de enfriamiento. Algunos como el logarítmico (Geman and Geman, 1984) garantizan una convergencia de manera casi absoluta, pero siendo equivalente a una búsqueda exhaustiva. En la práctica, se utilizan esquemas más rápidos, como el geométrico, que tiene la forma:

$$T^t = \mu T^{t-1} \tag{5.4}$$

para algún $0 < \mu < 1$.

5.1.2 Algoritmo de búsqueda

El algoritmo que se propone en este trabajo combina las ideas de muestreo y búsqueda provenientes de los algoritmos Metropolis-Hastings y

simulated annealing. La idea es recorrer los nodos del modelo, evaluando en la función objetivo elementos clasificados en dichos nodos, para así estimar la calidad de los nodos. De esta manera se realiza una exploración y a la vez una caracterización del modelo con respecto a la función objetivo.

En el contexto de esta aplicación, el concepto de estados para el algoritmo de Metropolis-Hastings sería equivalente a los nodos del modelo obtenido a partir de la aplicación del algoritmo GNGM. Así, la aplicación del algoritmo Metropolis-Hastings, generaría una secuencia de nodos según cierta función de distribución. En nuestro caso, se desea que se generen con más frecuencia aquellos nodos con un mejor valor de la calidad estimada del nodo. Por otra parte, a pesar de que las soluciones al problema práctico son los elementos, desde el punto de vista del empleo de SA, las soluciones son también los nodos.

El algoritmo parte del modelo obtenido a partir de la aplicación del algoritmo GNGM, o sea, un grafo con un conjunto de vértices (o nodos) $\mathcal{V}$ y un conjunto de aristas $\mathcal{A}$. Para un correcto funcionamiento del algoritmo se garantiza que este grafo sea conexo. Además, se tiene el conjunto de datos $\mathcal{D}$, que es el conjunto de datos poco estructurados que se encontraba en un banco de datos. Cada elemento $o^i \in \mathcal{D}$ está asociado a un nodo $v_i \in \mathcal{V}$ del modelo, según se clasificó anteriormente.

La principal particularidad del algoritmo que se propone (algoritmo 5.4) es el empleo de los estimadores $\hat{p}_i$ de los nodos como elemento que define el movimiento de un nodo a uno vecino. En este caso, el estimador que se emplea $\hat{p}_i$ para cada nodo v_i es la media aritmética de la evaluación de la función objetivo f para cada elemento o^i_k del conjunto de datos que pertenece a ese nodo:

$$\hat{p}_i = \frac{1}{n_i} \sum_{k=1}^{n_i} f(o^i_k) \tag{5.5}$$

donde n_i es la cantidad de elementos que pertenecen al nodo v_i.

El valor de los estimadores se va actualizando (y mejorando), pues en cada iteración se realiza una nueva evaluación de dos elementos, uno del nodo en que se encuentra la ejecución, y uno del nodo vecino seleccionado como posible transición. El algoritmo garantiza que no se repita

Algoritmo 5.4 Búsqueda basada en estimadores de calidad.

1. Seleccionar un nodo aleatorio v_i.
2. Seleccionar aleatoriamente un elemento o_k^i clasificado en el nodo v_i que no haya sido evaluado.
3. Evaluar $f(o_k^i)$, y actualizar el estimador de la calidad del nodo $\hat{p}_i$.
4. Actualizar óptimo global.
5. Seleccionar un nodo vecino $v_j = vecindad(v_i)$.
6. Seleccionar aleatoriamente un elemento o_l^j clasificado en el nodo v_j que no haya sido evaluado.
7. Evaluar $f(o_l^j)$, y actualizar el estimador de la calidad del nodo $\hat{p}_j$.
8. Actualizar óptimo global.
9. Si $min(1, e^{-\frac{\hat{p}_j - \hat{p}_i}{kT}}) > random(0,1)$, pasar al nodo v_j ($v_i^{t+1} = v_j^t$), si no permanecer en el nodo v_i ($v_i^{t+1} = v_i^t$).
10. Decrementar T según el esquema de enfriamiento.
11. Repetir los pasos del 2 al 10 hasta que se alcance la condición de parada.

la evaluación de un elemento y, por tanto, si todos los elementos pertenecientes a un nodo han sido evaluados, el estimador habrá alcanzado su valor final.

Una situación que puede presentarse, es que no existan elementos asociados a un nodo en particular. En este caso, el estimador de dicho nodo se toma como el promedio de los estimadores de los nodos vecinos, o sea:

$$\hat{p_i} = \frac{1}{|\mathcal{V}_i|} \sum_{j \in \mathcal{V}_i} \hat{p_j} \tag{5.6}$$

donde $|\mathcal{V}_i|$ es la cardinalidad del conjunto de los elementos $v_j \in \mathcal{V}$ para los cuales existe una arista $(v_i, v_j) \in \mathcal{A}$ en el modelo.

La condición de parada del algoritmo puede ser cuando se haya alcanzado una cantidad determinada de iteraciones o cuando se encuentre un valor significativamente pequeño de la función objetivo.

5.1.3 Resultados experimentales

En este epígrafe se exponen un grupo de experimentos orientados a mostrar la eficacia del algoritmo propuesto.

Experimentación en un conjunto de datos en tres dimensiones

El objetivo de este experimento es evaluar el algoritmo de búsqueda propuesto, dentro del procedimiento general ofrecido en la tesis, empleando un conjunto de datos de tres dimensiones. Este conjunto de datos produce un modelo donde es más fácil estudiar el funcionamiento del algoritmo propuesto.

El conjunto de datos (10 000 elementos) es similar al empleado en capítulos anteriores (*Dataset* 1 en la figura 3.3), pero al que le ha sido añadida una tercera dimensión (ver figura 5.1).

Para llevar a cabo el experimento, el conjunto de datos es procesado de manera similar a los procedimientos referidos en capítulos anteriores, como se muestra en la figura 5.2. En primer lugar, se emplea una representación de los elementos en dos dimensiones. El objetivo de esta simplificación es reflejar la pérdida de información que en general se

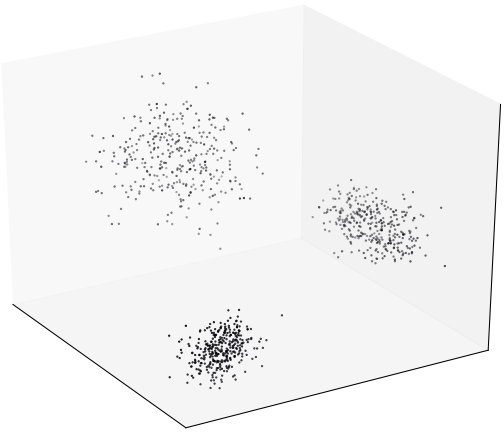


Figura 5.1 Conjunto de datos en tres dimensiones.

produce al emplear representaciones de los datos que son inexactas, pero necesarias para que puedan ser procesados, como ocurre cuando se aplica el método de grafos embebidos en espacios vectoriales.

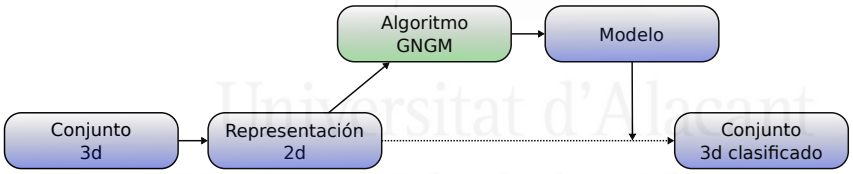
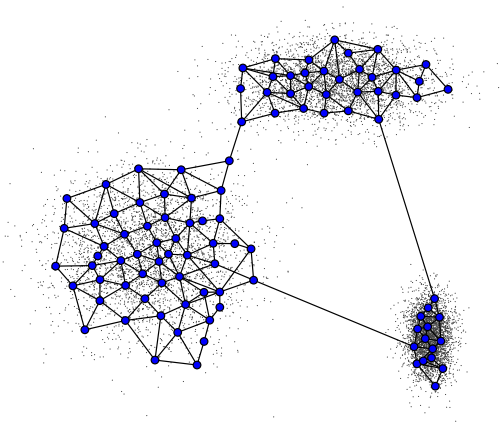


Figura 5.2 Diagrama del procedimiento de clasificación del conjunto de datos en tres dimensiones.

En segundo lugar, se aplica el algoritmo GNGM empleando dichas representaciones, obteniendo como resultado un modelo, como el que se muestra en la figura 5.3. A partir de este modelo, se clasifica cada una de las representaciones en dos dimensiones, obteniendo un conjunto clasificado, o sea, cada elemento del conjunto original está asociado a un nodo del modelo.



Parámetros GNGM ($\lambda = 50$, $ed_{max} = 50$, $\alpha = 0.5$, $\alpha' = 0.9$, $\varphi = 0.05$, $\gamma = 0.05$, $\sigma = 0.05$)

Figura 5.3 Conjunto de datos en dos dimensiones y modelo resultante de la aplicación del algoritmo GNGM.

En este experimento, lo que se define como función objetivo es encontrar el elemento más similar a un punto del espacio original. En particular, el punto del espacio original va a ser un elemento del conjunto, y por tanto, esa será la mejor solución. Por supuesto, la función objetivo está definida para las representaciones originales de los elementos (o sea, en tres dimensiones).

El procedimiento realizado se refleja en la figura 5.4. Se emplea el conjunto de datos en tres dimensiones clasificado, para aplicar el algoritmo de búsqueda propuesto. El algoritmo, como se refirió en epígrafes anteriores, realiza evaluaciones de la función objetivo, recorriendo el modelo generado en el preprocesamiento. Finalmente se obtiene como solución el elemento que se encuentra a menor distancia del punto objetivo.

En la figura 5.5 se observa el estado de la búsqueda en un punto intermedio de una ejecución del algoritmo. Los puntos negros pequeños son los elementos que han sido evaluados (su representación en dos dimensiones), en azul se representa el modelo de clasificación, y en rojo

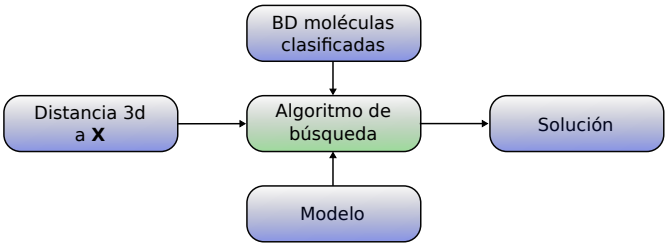


Figura 5.4 Diagrama del procedimiento experimental de búsqueda para el conjunto de datos de tres dimensiones.

el punto que constituye la mejor solución de la función objetivo. En la figura se puede apreciar cómo han sido evaluados elementos en una área considerable del espacio de búsqueda, pero que la mayor concentración se observa en la zona donde se encuentra la mejor solución. Este es precisamente el comportamiento deseado, pues refleja la efectividad del empleo de los estimadores de los nodos como guía de la heurística de búsqueda empleada.

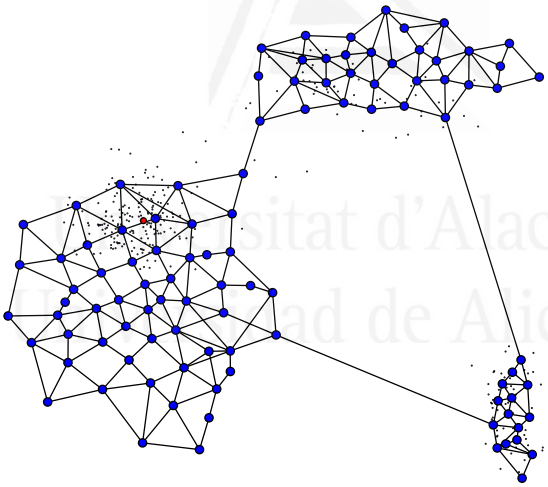


Figura 5.5 Elementos del conjunto de datos evaluados hasta un punto intermedio de una ejecución del algoritmo de búsqueda. Puntos negros: elementos evaluados. Punto rojo: solución óptima.

Los resultados del experimento se analizaron considerando dos enfoques. El primero es determinar la posición que ocupa la mejor solución encontrada hasta cierta iteración, en el *ranking* de las soluciones (siendo la mejor solución el 0 del *ranking*). El segundo enfoque, que es más intuitivo, tiene en cuenta el error de la mejor solución encontrada hasta cada iteración (o sea, la distancia a la mejor solución).

En muchas aplicaciones prácticas el segundo enfoque es más frecuente, pues en muchos casos es suficiente una solución con cierto margen de error. Sin embargo, el análisis de los resultados del algoritmo tomando en cuenta solo este enfoque puede ser engañoso, por ejemplo, en casos donde la solución óptima esté en una región con una alta densidad de soluciones. En este caso, una solución puede parecer muy buena desde el punto de vista del enfoque del error, pero no desde el punto de vista del *ranking* de las soluciones.

En la figura 5.6 se muestran los resultados generales de distintas ejecuciones del experimento. En el mismo se comparan los resultados del algoritmo propuesto, con los resultados que se obtienen utilizando el propio algoritmo pero con T en su valor máximo (o sea, comportándose como una búsqueda aleatoria en el modelo). Los gráficos muestran la mejor solución encontrada (en cuanto a orden o distancia) hasta cada iteración. En cada caso fueron combinadas las curvas correspondientes a cada ejecución independiente del algoritmo.

De manera general, se puede apreciar cómo en las primeras etapas el comportamiento es ligeramente similar, pero luego el algoritmo propuesto converge mucho más rápidamente a mejores soluciones. Alrededor de la iteración 400, el promedio de la mejor solución encontrada por el algoritmo promedio está alrededor de la solución 1 del *ranking* de las soluciones, con la evaluación del 4 % de los elementos del conjunto de datos. Cuando se ha evaluado el 7 % de los elementos, casi todas las búsquedas han encontrado la mejor solución.

Experimentación en un conjunto de datos de moléculas

El objetivo de este grupo de experimentos es comprobar la eficacia del algoritmo de búsqueda propuesto en un conjunto de datos con características similares a las que motivaron el origen de esta investigación.

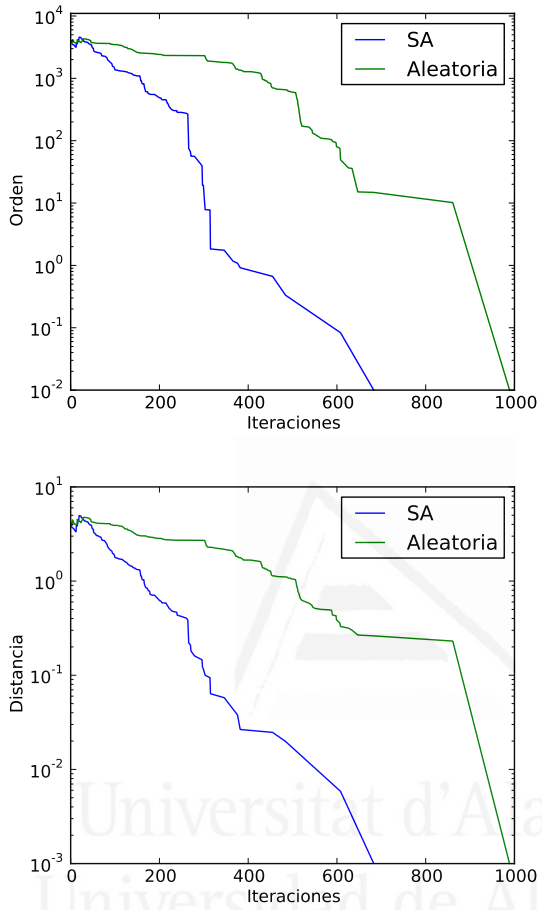


Figura 5.6 Comportamiento del *ranking* de la mejor solución encontrada (arriba) y de la distancia a la solución óptima (abajo) hasta cada iteración, entre el algoritmo de búsqueda propuesto y una búsqueda aleatoria.

El conjunto empleado es el mismo utilizado el capítulo anterior (11 347 moléculas), descrito en el epígrafe 4.2.1.

En la figura 5.7 se muestra el preprocesamiento que se realiza con el conjunto de experimentación. El preprocesamiento consiste en emplear el método de grafos embebidos en espacios vectoriales, para obtener una representación vectorial de cada elemento del conjunto de datos. Luego, a partir de estas representaciones y con el empleo del algoritmo GNGM, se obtiene un modelo que es utilizado para clasificar cada elemento del conjunto de datos.

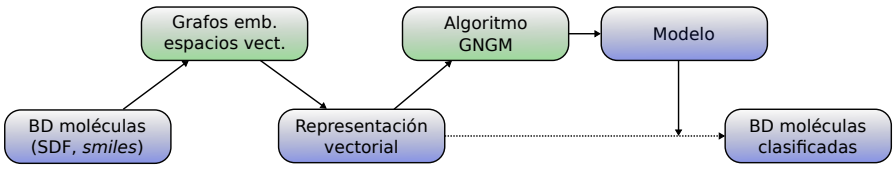


Figura 5.7 Diagrama del procedimiento de clasificación del conjunto de moléculas.

La experimentación se realiza para cuatro funciones objetivos distintas. De manera similar al experimento del subepígrafe anterior, la función objetivo es encontrar el elemento más cercano a un punto determinado. En este caso, se realizan experimentos independientes para cada una de las funciones objetivo.

Las funciones objetivo utilizadas son funciones de distancia entre los *fingerprints* de las moléculas. Las definiciones de distancias empleadas son: distancia de Tanimoto, distancia de Dice, distancia del coseno y distancia de Sokal (Willett et al., 1998; Lipkus, 1999) (ver tabla 5.1).

En la figura 5.8 se muestra el diseño general de los experimentos realizados. Aunque la función objetivo es distinta para cada experimento independiente, el algoritmo de búsqueda emplea el mismo modelo y la misma clasificación del conjunto de datos.

En la figuras 5.9 y 5.10 se muestran los resultados obtenidos en el proceso de experimentación. En la figura 5.9 se estudia el comportamiento del algoritmo de búsqueda con respecto a la mejor solución encontrada hasta una iteración. Las mejores soluciones encontradas por el algoritmo de búsqueda son analizadas desde el punto de vista del orden que ocupa

Tabla 5.1 Definiciones de las medidas de distancia para vectores binarios empleadas en la experimentación.

Distancia	**Fórmula**
Tanimoto	$1 - \frac{\lvert A \cap B \rvert}{\lvert A \rvert + \lvert B \rvert - \lvert A \cap B \rvert}$
Dice	$1 - \frac{2\lvert A \cap B \rvert}{\lvert A \rvert + \lvert B \rvert + 2\lvert A \cap B \rvert}$
Coseno	$1 - \frac{\lvert A \cap B \rvert}{\sqrt{(\lvert A \rvert + \lvert A \cap B \rvert) \times (\lvert B \rvert + \lvert A \cap B \rvert)}}$
Sokal	$1 - \frac{\lvert A \cap B \rvert}{\lvert A \cap B \rvert + 2(\lvert A \rvert + \lvert B \rvert - 2\lvert A \cap B \rvert)}$

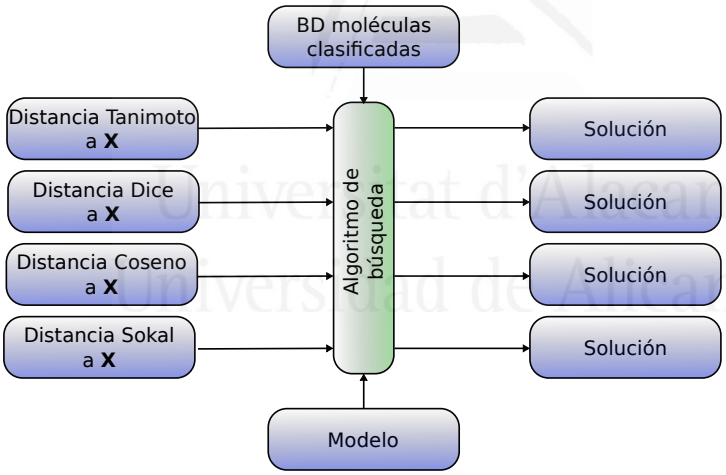


Figura 5.8 Diagrama del procedimiento experimental de búsqueda en el conjunto de datos de moléculas.

dicha solución en el *ranking* de todos los elementos del conjunto de datos con respecto a la función objetivo.

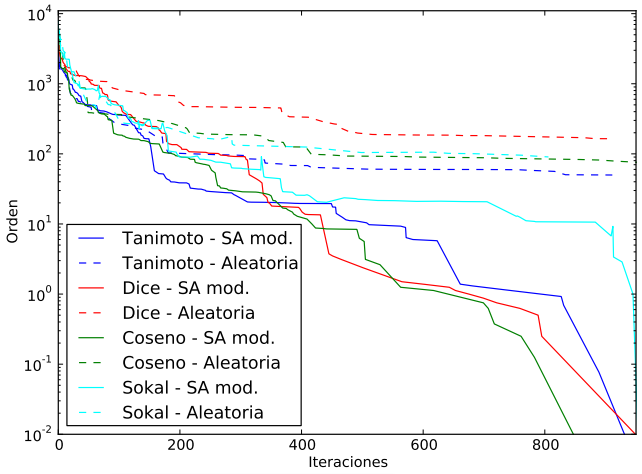


Figura 5.9 Comparación del *ranking* la mejor solución encontrada hasta cada iteración, entre el algoritmo de búsqueda y su respectiva búsqueda aleatoria, para cada función objetivo.

En la figura se aprecian los resultados combinados de varias ejecuciones del algoritmo para cada una de las funciones objetivos. Además, para establecer un criterio de comparación, se muestran los resultados de las búsquedas aleatorias respectivas para cada medida de distancia. Se puede notar que en todos los casos, el algoritmo de búsqueda propuesto se comporta significativamente mejor que la correspondiente búsqueda aleatoria. Además, en todos los casos se logra una solución relativamente buena (entre las 10 mejores soluciones) evaluando el 10 % de los elementos del conjunto de datos como mucho. A pesar de que los resultados para la distancia de Sokal son un poco peores, se puede decir que, en general, los resultados son bastante similares para cada uno de los procesos de búsqueda realizados.

En la figura 5.10 también se estudia el comportamiento del algoritmo de búsqueda con respecto a la mejor solución encontrada hasta una iteración, pero en este caso, se analiza el comportamiento con respecto

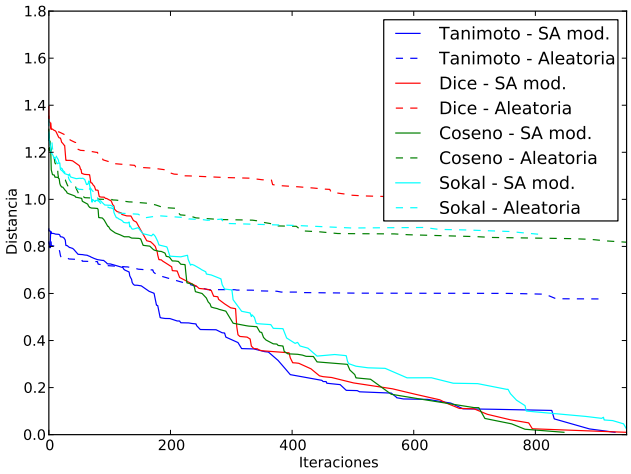


Figura 5.10 Comparación de la distancia al óptimo de la mejor solución encontrada hasta cada iteración, entre el algoritmo de búsqueda y su respectiva búsqueda aleatoria, para cada función objetivo.

a la distancia de la mejor solución encontrada a la mejor solución. Los resultados son similares a los que se observan con el enfoque del *ranking*. El algoritmo de búsqueda propuesto converge más rápidamente y encuentra mejores soluciones que una búsqueda aleatoria, y esto, con la evaluación de solo un 10 % de los elementos del conjunto de datos como mucho.

Los resultados obtenidos confirman la efectividad, no solo del algoritmo de búsqueda propuesto, sino de la solución general planteada en la investigación, pues a partir de una única organización del conjunto de datos se obtienen soluciones de calidad para varias funciones objetivo, realizando evaluaciones a un porcentaje relativamente bajo de los elementos del conjunto de datos.

5.2 Búsqueda en paralelo

El algoritmo de búsqueda propuesto en el epígrafe anterior puede ser mejorado desde el punto de vista del tiempo de respuesta, mediante la

implementación de variantes del mismo que aprovechen la capacidad de cómputo de múltiples unidades de procesamiento.

5.2.1 Algoritmo de búsqueda en paralelo

Una idea simple de aprovechamiento de múltiples unidades de procesamiento sería la ejecución simultánea del mismo proceso de búsqueda a partir de soluciones iniciales distintas. Esta idea, tiene como principal inconveniente que la evaluación de un mismo elemento del conjunto de datos puede repetirse en distintas búsquedas, lo cual constituye una pérdida de tiempo que puede ser considerable.

En la figura 5.11 se muestra un diseño general de un algoritmo de búsqueda en paralelo que elimina la repetición de evaluaciones. En cada unidad de procesamiento estará ejecutándose un proceso de búsqueda independiente tal como fue descrito en el epígrafe anterior. Cada instancia de la búsqueda obtiene los elementos del espacio de búsqueda de una fuente de datos externa común. Entre las instancias se intercambia información, y como resultado se obtienen dos beneficios fundamentales: evitar repetir la evaluación de un mismo elemento y mantener modelos con estimadores de más calidad.

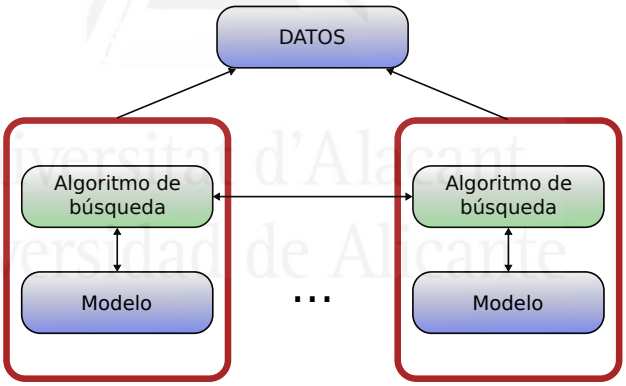


Figura 5.11 Diagrama del diseño general de la implementación de la búsqueda en paralelo.

Estos beneficios contribuyen directamente a la obtención de mejores resultados, tanto en calidad de las soluciones, como en tiempo de

respuesta. Por una parte, se evita la repetición de la evaluación de un mismo elemento en cada instancia, con lo que se reduce considerablemente el tiempo de respuesta. Además, se aprovechan las evaluaciones de elementos realizadas en otras instancias, lo que contribuye a mantener un modelo con estimaciones más precisas de la calidad de cada nodo, con el consiguiente beneficio para los procesos de búsquedas que se realizan en cada instancia.

En la figura 5.12 se muestra en detalle cómo se incorpora el mecanismo de comunicación entre las instancias de búsqueda que se ejecutan en paralelo. Para evitar que el tiempo asociado a la comunicación y procesamiento de mensajes afecte el tiempo de procesamiento de cada instancia de búsqueda, se crearon dos hilos de ejecución adicionales.

El primero de los hilos tiene la función de enviar las notificaciones al resto de las instancias tras la realización de una evaluación de un elemento. La información que se envía es qué elemento del conjunto de datos fue evaluado, cuál fue el resultado de la evaluación y en qué nodo se encuentra ubicado.

El segundo hilo tiene la función de recibir las notificaciones del resto de las instancias de búsqueda. Esta información se utiliza en el proceso de actualización del modelo, justo antes de que se utilicen las estimaciones de calidad de los nodos (en la evaluación de la condición de movimiento entre nodos).

La utilización de hilos de ejecución distintos para los mecanismos de notificación evita que sea necesario que todas las instancias de ejecución se encuentren en el mismo punto para que ocurra el intercambio de información. Este aspecto es importante pues los tiempos de evaluación de la función objetivo para un elemento del conjunto de datos son relativamente altos y, además, el tiempo de evaluación puede variar para distintos elementos.

5.2.2 Algoritmo de búsqueda en paralelo con intercambio de temperaturas

En el algoritmo paralelo propuesto en el epígrafe anterior se cuenta con múltiples instancias de búsqueda, cada una basada en la heurística SA,

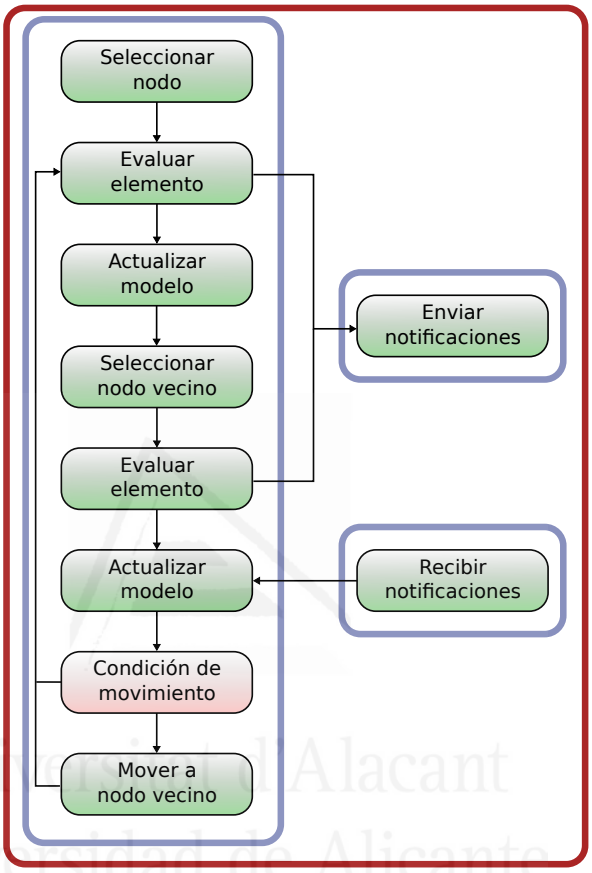


Figura 5.12 Diagrama del diseño general de una instancia del algoritmo de búsqueda en paralelo.

que comparten información sobre las evaluaciones de la función objetivo para elementos del conjunto de datos. En este epígrafe se propone un algoritmo paralelo basado en el algoritmo *parallel tempering*.

Parallel tempering

El algoritmo *parallel tempering* (PT) (Swendsen and Wang, 1986; Marinari and Parisi, 1992; Geyer and Thompson, 1995; Earl and Deem, 2005), también conocido como *replica exchange* MCMC, es un método de simulación que tiene como objetivo mejorar las propiedades dinámicas de los métodos de cadenas de Markov de Monte Carlo.

Esencialmente, PT consiste en ejecutar múltiples réplicas de la simulación de un sistema, bajo diferentes condiciones termodinámicas, generalmente definidas por la temperatura. Las réplicas a altas temperaturas son capaces de explorar grandes regiones del espacio de búsqueda, mientras que las réplicas a bajas temperaturas se enfocan en la exploración de regiones específicas.

Durante el proceso de simulación, las réplicas con niveles de temperaturas adyacentes pueden intercambiar sus configuraciones en determinados momentos, sujeto a un criterio de aceptación. Esto permite que configuraciones a altas temperaturas estén disponibles a simulaciones a bajas temperaturas.

El criterio de aceptación de intercambio de configuraciones que se emplea, se basa en el criterio de actualización del algoritmo Metropolis-Hastings. Dadas dos réplicas a temperaturas T_1 y T_2, se realiza un intercambio de configuraciones con probabilidad:

$$p = \text{mín}\left(1, e^{-(E(s_j)-E(s_i))\left(\frac{1}{kT_j}-\frac{1}{kT_i}\right)}\right) \tag{5.7}$$

o sea, se realiza un intercambio cuando la réplica a mayor temperatura tiene una energía inferior a la réplica a menor temperatura.

En la mayor parte de los casos es computacionalmente más sencillo y eficiente intercambiar la temperatura que las configuraciones de cada réplica.

Mediante una adecuada selección de las temperaturas y del número de réplicas, PT puede relajar el tiempo de una simulación de Monte

Carlo en un sistema físico, y también puede mejorar la convergencia a un mínimo global (Rathore et al., 2005). Así, el uso de PT es más conveniente que la ejecución de múltiples MCMC sin intercambio entre las réplicas.

Algoritmo de búsqueda basado en PT

En la figura 5.13 se muestra la adaptación de la idea del algoritmo PT a la propuesta de búsqueda en paralelo. El esquema general con respecto al algoritmo de búsqueda en paralelo basado en SA se mantiene: en cada unidad de procesamiento se realizan búsquedas independientes, que se intercambian información.

La principal modificación que se realiza en el hilo de búsqueda, con respecto a la propuesta anterior, es la incorporación de etapas para realizar una solicitud de intercambio y para actualizar el valor de la temperatura. En este caso, a diferencia del algoritmo basado en SA, la modificación de las temperaturas ocurre a partir del intercambio entre instancias de búsqueda. Otra modificación es que en cada instancia de búsqueda se añade un nuevo hilo que es el encargado de llevar la lógica del intercambio de temperaturas.

En muchas implementaciones de PT, los intentos de intercambio de temperatura entre procesos a temperaturas adyacentes se realizan cada cierto número de iteraciones. Sin embargo, en la mayoría de las aplicaciones prácticas, no todas las instancias de búsqueda alcanzan el punto de intercambio al mismo tiempo, entre otras causas por la variación del tiempo de evaluación de las funciones objetivos (Li et al., 2009). De esta manera, el primer proceso que alcance este punto debe esperar al otro, tal vez por un período largo de tiempo. Por esta razón, en el algoritmo que se propone se utiliza un esquema asíncrono, donde no se espera por la respuesta para hacer el intercambio. Si posteriormente no se aceptara el intercambio (lo cual es, en general, bastante probable) la búsqueda continúa sin desperdicio de tiempo. Si posteriormente el intercambio fuera aceptado, el procesamiento realizado ha contribuido a aprovechar el tiempo computacional en la evaluación de elementos del conjunto de datos, mejorando los estimadores de calidad de los nodos del modelo.

En el algoritmo propuesto, las solicitudes de intercambio de tempe-

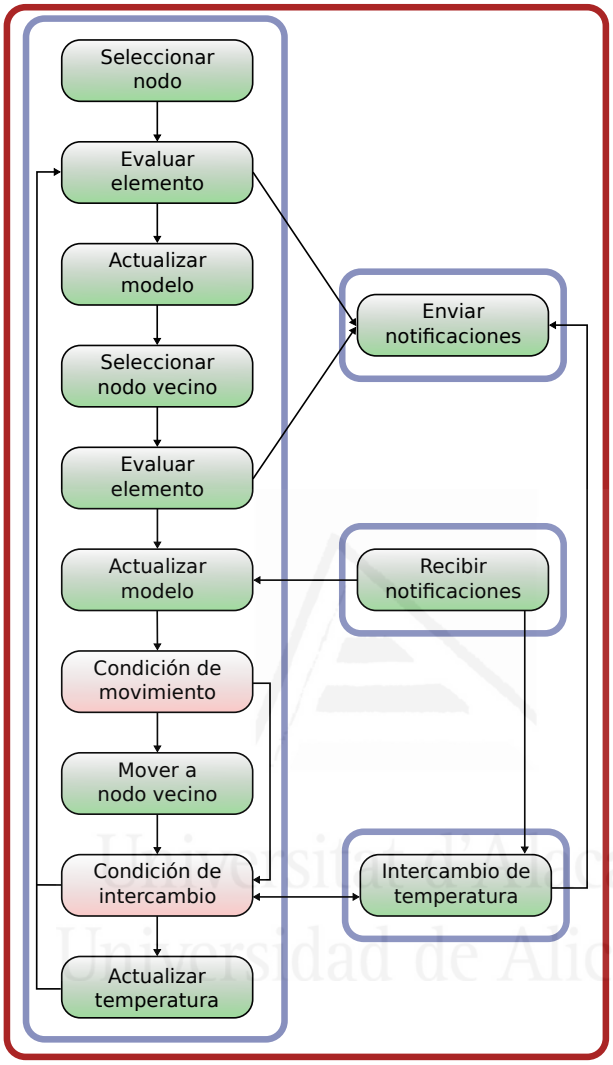


Figura 5.13 Diagrama del diseño general de una instancia del algoritmo de búsqueda en paralelo con la variante del *parallel tempering*.

ratura se realizan cada cierto número de iteraciones, a través del hilo independiente que maneja la lógica del intercambio de temperatura entre instancias de búsquedas independientes. De esta manera, una vez que se realiza una solicitud, el proceso de búsqueda no se detiene a esperar un resultado, sino que continúa funcionando.

Los mensajes que se envían entre distintas instancias de búsqueda, utilizan los hilos de comunicación ya establecidos para el envío y recepción de mensajes.

Cuando se recibe una aceptación de intercambio se procede a actualizar la temperatura en la instancia actual, volviendo además, al nodo en que se encontraba la búsqueda en el momento de realizar la solicitud de aceptación. Esta información se almacena cada vez que se realiza una solicitud.

En la instancia de búsqueda donde se recibe una solicitud de intercambio se realiza la evaluación de la condición de aceptación. Si se cumple la condición se envía un mensaje de aceptación y se espera por una confirmación. En caso de cumplirse la condición se envía un mensaje de rechazo del intercambio. Este procedimiento también se realiza en el hilo que controla el intercambio de temperatura, por lo que tampoco se detiene el proceso de búsqueda.

5.2.3 Resultados experimentales

En este epígrafe se detallan el diseño y los resultados de los experimentos realizados para ilustrar el funcionamiento de los algoritmos paralelos propuestos, así como para comprobar la efectividad de sus resultados.

Experimentos para ilustrar el funcionamiento del algoritmo de búsqueda con PT

Este grupo de experimentos está orientado a ilustrar el funcionamiento de la variante del algoritmo propuesto que emplea la idea del algoritmo PT.

Para estos experimentos se utilizó el mismo conjunto de datos empleado en el subepígrafe 5.1.3 (10 000 elementos en tres dimensiones). Así mismo, se realizó el mismo preprocesamiento de los datos descrito

en la figura 5.2 y el mismo procedimiento experimental reflejado en la figura 5.4.

En la figura 5.14 se muestra el comportamiento de la búsqueda de manera independiente para cada instancia de los dos algoritmos de búsqueda en paralelo propuestos. En el caso del algoritmo que se basa en SA se observa que cada instancia de búsqueda independiente converge a la región donde los nodos presentan valores de calidad mejores. Este es, en general, el comportamiento normal que se espera de una búsqueda.

En el caso del comportamiento de las distintas instancias para el algoritmo en paralelo basado en PT, se observa un comportamiento similar para algunas de las instancias, mientras otras tienen un comportamiento más irregular. Precisamente, las instancias que no convergen, realizan una exploración permanente del espacio de búsqueda mientras mantienen altos valores de la variable temperatura, y no quedan atrapadas en mínimos locales. Así, pueden proveer a las instancias que operan a bajas temperaturas de regiones con soluciones más difíciles de encontrar.

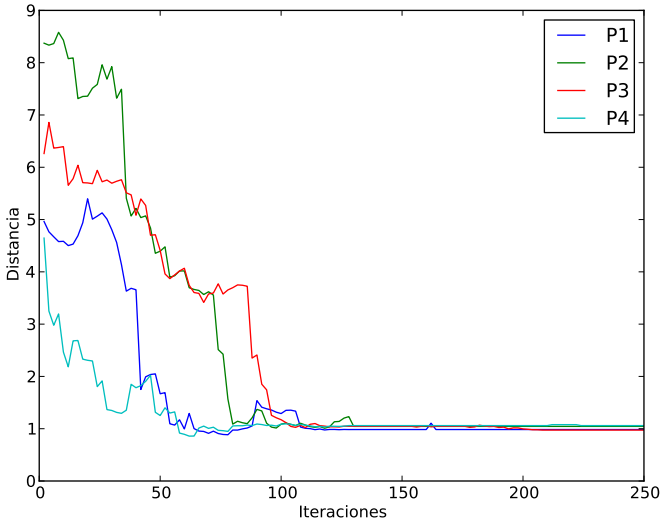
El comportamiento descrito anteriormente se puede observar con más claridad en la figura 5.15, pero a nivel de distintas temperaturas. Aquí se puede apreciar como, generalmente, las instancias que operan a temperaturas superiores se mantienen en regiones de soluciones peores, pues de entrar en una región con mejores valores de calidad, posiblemente intercambiará su temperatura con la instancia adyacente inferior en temperatura.

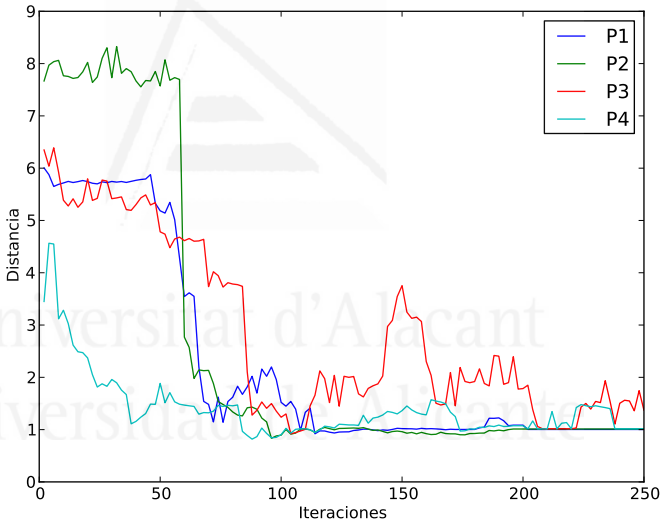
Finalmente, en la figura 5.16 se observa como se producen los cambios de temperatura entre los distintas instancias cuando tienen temperaturas adyacentes.

Comparación del comportamiento de las variantes de búsqueda en paralelo

El objetivo de este grupo de experimentos es comparar los resultados de los tres algoritmos de búsquedas en paralelo considerados: con búsqueda independientes basadas en SA, algoritmo paralelo basado en SA y algoritmo en paralelo basado en PT.

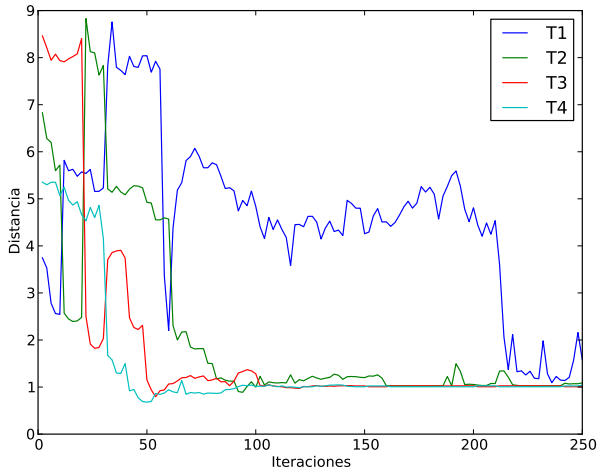
Para un primer análisis se utilizó el mismo conjunto de datos empleado en el subepígrafe 5.1.3 (10 000 elementos en tres dimensiones).





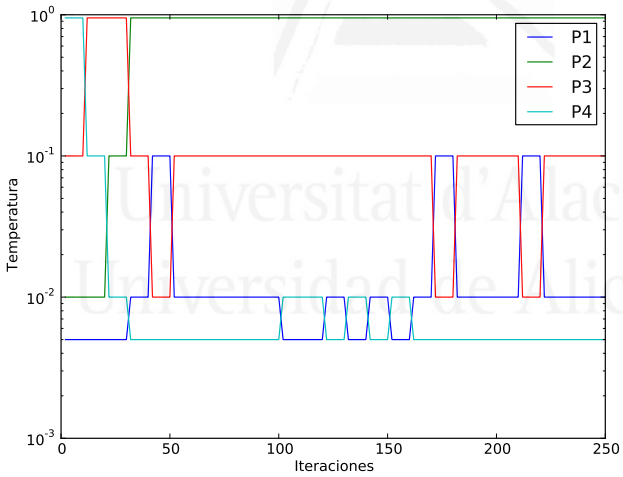
P1, P2, P3 y P4 son los distintos procesos paralelos.

Figura 5.14 Comparación del valor de la calidad del nodo en que se encuentra cada instancia de búsqueda para el algoritmo basado en SA (arriba) y el algoritmo basado en PT (abajo).



$T1 = 0.95$, $T2 = 0.1$, $T3 = 0.01$, $T4 = 0.005$

Figura 5.15 Comparación del valor de la calidad del nodo en que se encuentra cada instancia de búsqueda del algoritmo paralelo basado en PT, para temperaturas distintas.



P1, P2, P3 y P4 son los distintos procesos paralelos.

Figura 5.16 Intercambio de temperaturas entre instancias de búsqueda del algoritmo basado en PT.

Así mismo, se realizó el mismo preprocesamiento de los datos descrito en la figura 5.2 y el mismo procedimiento experimental reflejado en la figura 5.4.

En la figura 5.17 se observan los resultados de estos experimentos, tanto desde el punto de vista de la distancia como del orden de cada solución. Se puede apreciar como los mejores resultados se obtienen para la variante del algoritmo basada en PT, seguido por la variante basada en SA con actualización del modelo (PSA) y por último la variante de SA independientes (MSA).

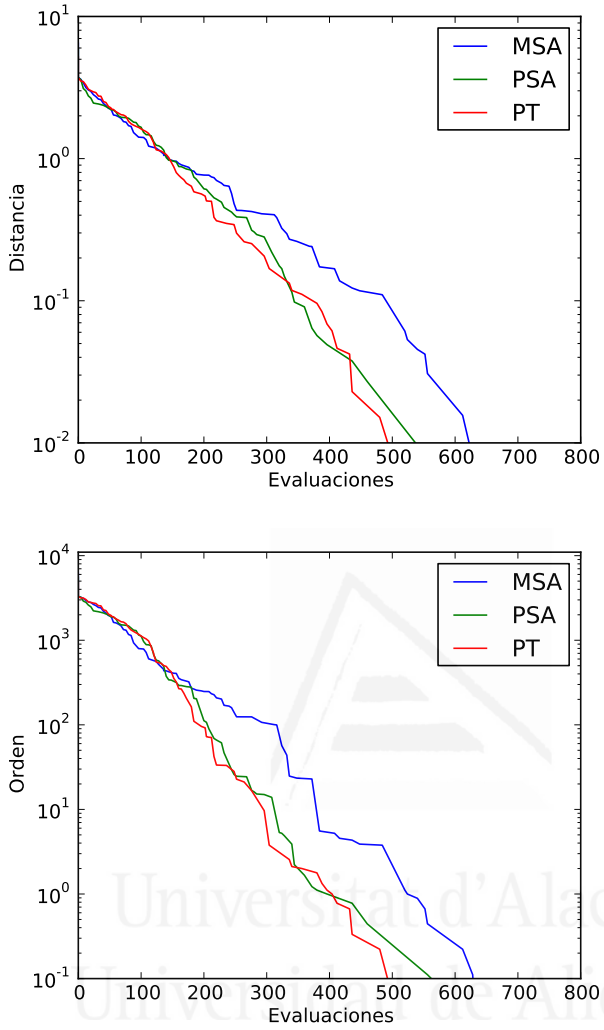
El hecho de que las variantes que intercambian información para actualizar sus modelos (PSA y PT) tengan mejores resultados es consistente con la idea de que tener modelos más precisos garantiza una búsqueda más precisa en cada instancia individual.

Los resultados de los algoritmos paralelos basados en SA y PT son bastantes similares, con una ligera mejoría del algoritmo PT. Este hecho se pudiera explicar teniendo en cuenta que la ventaja del algoritmo PT sobre SA es más clara en condiciones donde existe una estructura más compleja de las soluciones, donde la mayoría de las instancias de SA quedarían atrapadas en mínimos locales.

Para un segundo análisis se utilizó el mismo conjunto de datos empleado en el subepígrafe 4.2.1 (11 347 moléculas). Así mismo, se realizó el mismo preprocesamiento de los datos descrito en la figura 5.7 y el mismo procedimiento experimental reflejado en la figura 5.8, o sea, empleando las mismas cuatro funciones objetivo.

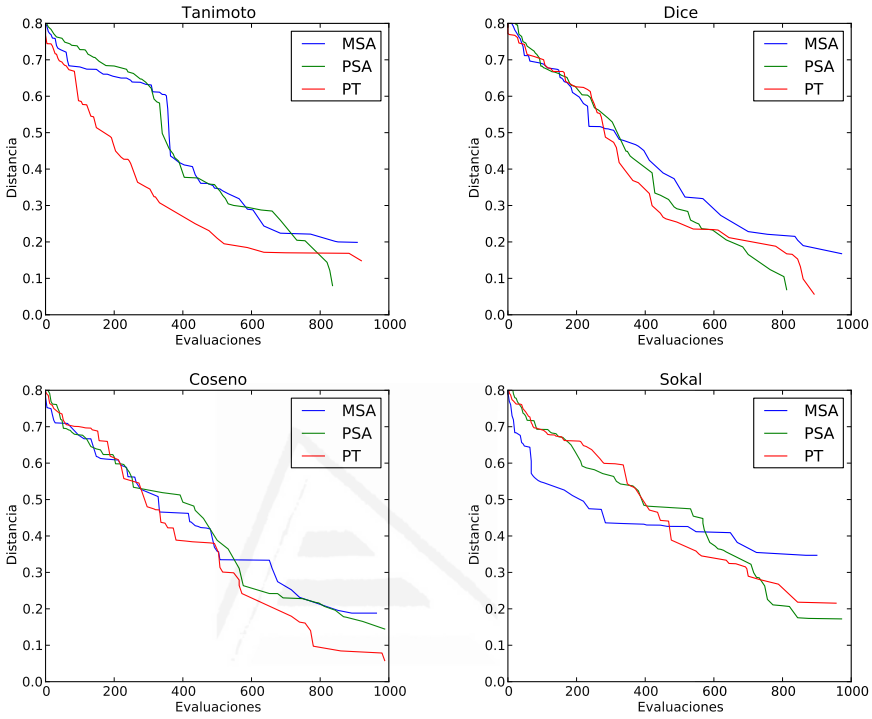
Los resultados se muestran en las figuras 5.18 y 5.19. Estos resultados son consistentes con los resultados obtenidos anteriormente para el conjunto de datos de tres dimensiones, o sea, una superioridad manifiesta de las variantes con intercambio de información para la actualización del modelo, así como una ligera superioridad de la variante PT sobre la variante MSA.

Este conjunto de experimentos permite confirmar que el intercambio de información que se realiza en las variantes de búsqueda en paralelo propuestas, contribuye a mejorar la calidad de las búsquedas con respecto a distintas funciones objetivo.



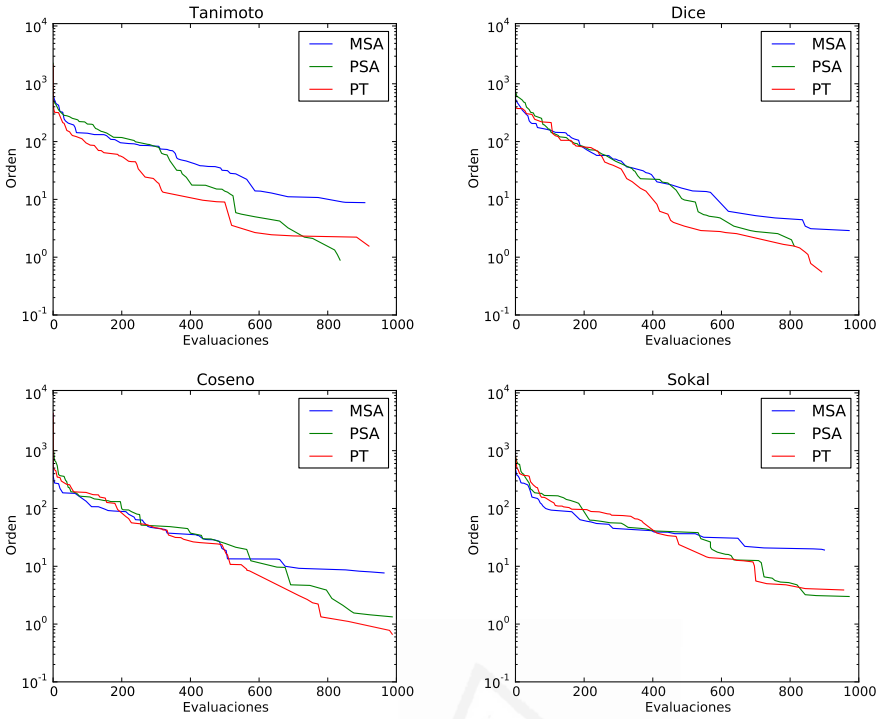
MSA: múltiples instancias de SA independientes
PSA: instancias de SA que intercambian información
PT: *parallel tempering*

Figura 5.17 Comparación de la distancia a la solución óptima (arriba) y el *ranking* (abajo) de la mejor solución encontrada por cada algoritmo de búsqueda con respecto a la cantidad de evaluaciones realizadas.



MSA: múltiples instancias de SA independientes
PSA: instancias de SA que intercambian información
PT: *parallel tempering*

Figura 5.18 Comparación de la distancia a la solución óptima de la mejor solución encontrada por cada algoritmo de búsqueda con respecto a la cantidad de evaluaciones realizadas, para cada función objetivo.



MSA: múltiples instancias de SA independientes
PSA: instancias de SA que intercambian información
PT: *parallel tempering*

Figura 5.19 Comparación del *ranking* de la mejor solución encontrada por cada algoritmo de búsqueda con respecto a la cantidad de evaluaciones realizadas, para cada función objetivo.

5.3 Sumario

En este capítulo se propone, primeramente, un algoritmo de búsqueda que basa su funcionamiento en la exploración de un espacio de búsqueda organizado según un modelo obtenido a partir de la utilización de un algoritmo de *clustering* con preservación de la topología. Este algoritmo se basa en la heurística *simulated annealing*, y la búsqueda se realiza sobre los nodos del modelo referido, tomando en cuenta una estimación de la calidad de los nodos con respecto a la función objetivo. Estas estimaciones se van calculando y actualizando a medida que se recorre

el modelo.

Posteriormente, se ofrece una variante del algoritmo en paralelo, donde en cada unidad de procesamiento se ejecuta una instancia de búsqueda, y las instancias intercambian información entre ellas, para tener modelos más precisos y evitar el análisis repetido de elementos. Además, se propone otro algoritmo en paralelo basado en el algoritmo *parallel tempering*.

Los resultados experimentales son coherentes con la idea del empleo de un estimador de la calidad de los nodos como guía de la búsqueda, y además sugieren una relativa superioridad del método de búsqueda en paralelo basado en el algoritmo *parallel tempering*.

Capítulo 6

Conclusiones

En esta investigación se propone un método para realizar búsquedas en grandes conjuntos de datos con respecto a una función objetivo. El método está dividido en dos etapas: la organización del conjunto de datos mediante el empleo de un algoritmo de *clustering* con preservación de la topología y la búsqueda de elementos que minimicen distintas funciones objetivo, según el modelo organizativo propuesto. El método se pone a prueba mediante la implementación de una propuesta de solución que abarca todas sus etapas.

La organización del conjunto de datos se lleva a cabo mediante el empleo de un algoritmo basado en el método *Growing Neural Gas*, al que se la han realizado modificaciones para adaptarlo al contexto específico de la investigación. Para permitir la aplicación del algoritmo sobre un conjunto de datos de grafos se emplea el método de grafos embebidos en espacios vectoriales, mediante el cual se logra una representación vectorial de los grafos. A partir del proceso de organización del conjunto de datos se obtiene un modelo que se utiliza para clasificar cada uno de los elementos del conjunto en uno de sus nodos. En este modelo existen relaciones de vecindad entre los nodos, que reflejan a su vez relaciones de similitud de los elementos que tienen asociados.

Para la etapa de búsqueda se ha diseñado un algoritmo basado en la heurística *simulated annealing*. El algoritmo recorre los nodos del modelo realizando evaluaciones de la función objetivo de elementos asociados a estos nodos. A partir de estas evaluaciones se realiza una estimación de

la calidad de los elementos en cada nodo, que se utiliza para orientar el recorrido de la búsqueda. También se ofrece una variante de búsqueda basada en el algoritmo *parallel tempering*.

Cada etapa de la propuesta de solución ha sido diseñada teniendo en cuenta el coste computacional que implica el procesamiento de grandes volúmenes de datos y, por tanto, se han implementado soluciones paralelas.

La viabilidad de cada una de las etapas de la propuesta de solución es validada mediante la realización de experimentos utilizando un grupo de conjuntos de datos y funciones objetivo. Los resultados más relevantes obtenidos en esta investigación muestran que el proceso de búsqueda de soluciones con respecto a un objetivo específico requiere, como promedio, que sean evaluados en la función objetivo menos de un 10 % de los elementos de los conjuntos de datos. De manera general, el conjunto de experimentos realizados en esta investigación validan de forma preliminar la posibilidad de emplear la solución propuesta en problemas reales más complejos.

6.1 Aportaciones

Las principales aportaciones resultantes de esta investigación son las siguientes:

- Se ha desarrollado un método general para optimizar búsquedas en grandes conjuntos de datos con características similares a los grafos.
- Se ha analizado el método de grafos embebidos en espacios vectoriales para comprobar si resulta factible su aplicación al problema planteado en este trabajo.
- Se propone una solución basada en el algoritmo *Growing Neural Gas*, que permite la obtención de modelos adaptados a condiciones específicas, que sirvan de apoyo al proceso de búsqueda.
- Se ha desarrollado un algoritmo de búsqueda basado en la estimación de la calidad de los nodos del modelo sobre el que opera.

- Se han implementado soluciones paralelas de los algoritmos propuestos para reducir el coste computacional de las diferentes partes del método general propuesto.

- Se ha realizado un conjunto de experimentos para validar cada uno de los métodos planteados en este trabajo.

6.2 Líneas de trabajo futuras

Esta investigación tuvo entre los puntos iniciales de la motivación una problemática que aparece durante el proceso de diseño de los medicamentos. El problema consiste en encontrar un compuesto químico, de entre la gran cantidad que existe, que presente las características adecuadas con respecto a un objetivo biológico específico. En este sentido, una clara línea de investigación futura es:

- Aplicar la solución propuesta en la resolución de problemas reales específicos, lo cual requerirá el empleo de representaciones más precisas de las moléculas.

Además, otras líneas de trabajo que se proyectan a partir de la presente investigación son:

- Estudiar qué nivel de detalle del modelo que se obtiene en la organización de los datos ofrece un equilibrio óptimo entre coste computacional y eficiencia en las búsquedas.

- Estudiar el empleo de procedimientos distintos en la etapa de organización, desde otras técnicas de *clustering* con preservación de la topología, hasta otros métodos de *clustering* que requieran un procesamiento adicional para lograr una estructura adecuada para la etapa de búsqueda.

- Estudiar la posibilidad de emplear métodos alternativos de organización del conjunto de datos, que eviten tener que utilizar formas de representación de los datos donde se pueda perder información.

Listado de acrónimos

BIRCH *Balanced Iterative Reducing and Clustering using Hierarchies*

BKT Burkhard-Keller *Tree*

BPS *Border Prototype Selector*

CHL *Competitive Hebbian Learning*

CLARA *Clustering Large Applications*

CLARANS *Clustering Large Applications based upon RANdomized Search*

DBSCAN *Density-Based Spatial Clustering of Applications with Noise*

GCS *Growing Cell Structures*

GHSOM *Growing Hierarchical Self Organizing Maps*

GNG *Growing Neural Network*

GNGM *Growing Neural Network* modificado

GNGME *Growing Neural Network* con modificación basada en el error

MAM Métodos de Acceso Métrico

MCMC *Markov Chain Monte Carlo*

MDS *MultiDimensional Scaling*

MPP *Massively Parallel Processors*

MST *Minimal Spanning Tree*

NG *Neural Gas*

OPTICS *Ordering Points To Identify the Clustering Structure*

PAM *Partition Around Medoids*

PT *Parallel Tempering*

SA *Simulated Annealing*

SAHN *Sequential Agglomerative Hierarchical Non-overlapping*

SMP *Symetric MultiProcessors*

SOM *Self Organizing Maps*

SPS *Spanning Prototype Selector*

VPT *Vantage-Point Tree*

Referencias

Abbass, H. A. (2002). From evolution to immune to swarm to ...? a simple introduction to modern heuristics. In *Data Mining: A Heuristic Approach*, pages 1–21. Idea Group Publishing.

Abbass, H. A., Sarker, R. A., and Newton, C. S. (2002). *Data Mining: A Heuristic Approach*. Idea Group Publishing.

Abello, J., Pardalos, P., and Resende, M. G. C. (2002). *Handbook of Massive Data Sets*. Springer-Verlag.

Aggarwal, C. C. and Wang, H. (2010). *Managing and mining graph data*. Springer.

Alahakoon, D., Halgamuge, S. K., and Srinivasan, B. (2000). Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, pages 601–614.

Altingovde, I. S., Engin, D., Fazli, C., and Özgür Ulusoy (2008). Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *ACM Transactions on Information Systems*, pages 15:1–15:36.

Amdahl, G. (1967). Validity of the single processor approach to achieving large-scale computing capabilities. *AFIPS*, pages 483–485.

Ansari, N. and Hou, E. (1997). *Computational intelligence for optimization*. Kluwer Academic Publisher.

Arroyave, G., Ortega, O., and Marín, A. (2002). A parallel implementation of the som algorithm for visualizing textual documents in a 2d plane.

Ayadi, T., Hamdani, T. M., Alimi, A. M., and Habou, M. A. (2007). 2ibgsom: interior and irregular boundaries growing self-organizing maps. *Sixth International Conference on Machine Learning and Applications*, pages 387–392.

Bahuer, H. U. and Villmann, T. (1997). Growing a hypercubical output space in a self- organizing feature map. *IEEE Transactions on Neural Networks*, pages 218–226.

Baker, M. (2001). Cluster computing. *White paper, University of Portsmouth*.

Baldi, P. and Brunak, S. (2001). *Bioinformatics: The Machine Learning Approach*. The MIT Press.

Baurin, N., Aboul-Ela, F., Barril, X., Davis, B., Drysdale, M., Dymock, B., Finch, H., Fromont, C., Richardson, C., Simmonite, H., and Hubbard, R. E. (2004). Design and characterization of libraries of molecular fragments for use in nmr screening against protein targets. *Journal of Chemical Information and Computer Sciences*, 44(6):2157–2166.

BioProject (2012). http://www.ncbi.nlm.nih.gov/bioproject.

BioTech FYI Center (2012). http://biotech.fyicenter.com/resource/database.html.

Blake, G., Dreslinski, R. G., and Mudge, T. (2009). A survey of multicore processors. *IEEE Signal Processing Magazine*, pages 26–37.

Bunke, H. and Neuhaus, M. (2007). Graph matching-exact and error-tolerant methods and the automatic learning of edit costs. In *Mining graph data*, pages 17–32. Wiley.

Burkhard, W. A. and Keller, R. M. (1973). Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236.

Cantú-Paz, E. and Kamath, C. (2002). On the use of evolutionary algorithms in data mining. In *Data Mining: A Heuristic Approach*, pages 48–71. Idea Group Publishing.

Carpenter, G. A. and Grossberg, S. (2003). Adaptive resonance theory. In *The Handbook of Brain Theory and Neural Networks*, pages 87–90. Cambridge MA: MIT Press.

Cavalli, E., Poluzzi, F., Ponti, M. D., and Recanatini, J. (2002). Compounds with herg k+ channel blocking activity. *Journal of Medical Chemistry*, 45:2844–2853.

Chaudhary, V., Feng, L., Matta, V., and Yang, L. T. (2006). Parallel implementations of local sequence alignment: hardware and software. In *Parallel computing for bioinformatics and computational biology*, pages 233–264. Wiley.

Chawla, S. (2008). New frontiers in applied data mining. In *Lecture Notes in Artificial Intelligence. PAKDD International Workshops*, pages Springer–Verlag.

Chen, X., III, A. R., and Young, S. S. (1998). Recursive partitioning analysis of large structure activity data sets using three-dimensional descriptors. *Journal of Chemical Information and Computer Sciences*, pages 1054–1062.

Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. (2001). Proximity searching in metric spaces. *ACM Computing Surveys*, 3(33):273–312.

Ciaccia, P., Patella, M., and Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. *The VLDB Journal*, page 426–435.

CIGB (2012). http://www.cigb.edu.cu.

Clarkson, K. L. (2006). Nearest-neighbor searching and metric space dimensions. In *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–79. MIT Press.

Conte, D., Foggia, P., Sansone, C., and Vento, C. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, pages 265–298.

Cook, D. J. and Holder, L. B. (2000). Graph-based data mining. *IEEE Intelligent Systems*, pages 32–41.

Cook, D. J. and Holder, L. B. (2007). *Mining Graph Data*. Wiley.

Cramer, R. D., Jilek, R. J., Guessregen, S., Clark, S. J., Wendt, B., and Clark, R. D. (2004). "lead hopping". validation of topomer similarity as a superior predictor of similar biological activities. *Journal of Medicinal Chemistry*, 47(27):6777–6791.

Dalby, A., Nourse, J. G., Hounshell, W. D., Gushurst, A. K. I., Grier, D. L., Leland, A., and Laufer, J. (1992). Description of several chemical structure file formats used by computer programs developed at molecular design limited. *Journal of Chemical Information and Computer Sciences*, 32:244–255.

de Castro, L. N. and von Zuben, F. J. (2002). Artificial immune systems: Using the immune system as inspiration for data mining. In *Data Mining: A Heuristic Approach*, pages 209–230. Idea Group Publishing.

Delaney, J. S. (2004). Aequeous solubility data for 1144 low molecular weight compounds. *Journal of Chemical Information and Computer Sciences*, 44:1000 – 1005.

Dhillon, I. S. and Modha, D. S. (1999). A data clustering algorithm on distributed memory multiprocessors. *KDD*.

Downs, G. and Barnard, J. (2002). Clustering methods and their uses in computational chemistry. *Reviews in Computational Chemistry*.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification 2nd Edition*. Wiley Sons.

Dutta, D. and Chen, T. (2007). Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search. *Bioinformatics*, pages 612–618.

Earl, D. J. and Deem, M. W. (2005). Parallel tempering: theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, 7:3910–3916.

Earth System Research Laboratory (2012). http://www.esrl.noaa.gov/psd/data/gridded/.

Eckert, H. and Bajorath, J. (2006). Design and evaluation of a novel class-directed 2d fingerprint to search for structurally diverse active compounds. *Journal of Chemical Information and Modeling*, 46(6):2515–2526.

Einakian, S. and Ghanbari, M. (2006). Parallel implementation of association rule in data mining. In *Thirty-Eighth Southeastern Symposium on System Theory*, pages 21–26.

Estivil-Castro, V. (2002). Approximating proximity for fast and robust distance- based clustering. In *Data Mining: A Heuristic Approach*, pages 22–47. Idea Group Publishing.

European Bioinformatics Institute (2012). http://www.ebi.ac.uk/Databases/.

Expert Health Data Programming (2012). http://www.ehdp.com/vitalnet/datasets.htm.

Feng, J., Lurati, L., Ouyang, H., Robinson, T., Wang, Y., Yuan, S., and Young, S. S. (2003). Predictive toxicology: benchmarking molecular descriptors and statistical methods. *Journal of Chemical Information and Computer Sciences*, 43(5):1463–1470.

Filho, R. F. S., Traina, A. J. M., and Faloutsos, C. (2001). Similarity search without tears: The omni family of all-purpose access methods. In *International Conference on Data Engineering ICDE01*, pages 623–630.

Flower, D. R. (1998). On the properties of bit string-based measures of chemical similarity. *Journal of Chemical Information and Computer Sciences*.

Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Transaction on Computers*, page 948.

Foti, D., Lipari, D., Pizzuti, C., and Talia, D. (2000). Scalable clustering for data mining on multicomputers. In *High Performance Data Mining Workshop*.

Fritzke, B. (1995). A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632.

Furao, S. and Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, pages 90–106.

Furao, S. and Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks*, pages 893–903.

Gancev, S. and Kulakov, A. (2009). Modified growing neural gas algorithm for faster convergence on signal distribution sudden change. In *XXII International Symposium on Information, Communication and Automation Technologies (ICAT '09)*, pages 1– 5.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

Geyer, C. J. and Thompson, E. A. (1995). Annealing markov chain monte carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90:909–920.

Gilks, W., Richardson, S., and Spiegelhalter, D. (1996). *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*. Interdisciplinary Statistics Series. Chapman & Hall.

Grossman, R. L., Kamath, C., Kegelmeyer, P., Kumar, V., and Nambury, R. (2001). *Data Mining for Scientific and Engineering Applications*. Springer-Verlag.

Guha, S., Rastogi, R., and Shim, K. (1998). Cure: An efficient clustering algorithm for large databases. In *Proceedings of ACM-SIGMOD International Conference of Management of Data*, pages 73–84.

Hadjidoukas, P. and Amsaleg, L. (2008). Parallelization of a hierarchical data clustering algorithm using openmp. *Lecture Notes in Computer Science (Springer)*, pages 289–299.

Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques. 2nd edition*. Morgan Kaufmann Publishers.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:97–109.

Hebboul, A., Hacini, M., and Hachouf, F. (2011). An incremental parallel neural network for unsupervised classification. In *7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA)*.

Hubert, L. and Schultz, J. (1976). Quadratic assignment as a general data analysis strategy. *British Journal of Mathematical and Statistical Psychology*, 29(2):190–241.

Human Genome Project (2012). http://www.ornl.gov/sci/techresources/HumanGenome.

Huuskonen, J. (2000). Aequeous solubility data for 1302 low molecular weight compounds. *Journal of Chemical Information and Computer Sciences*, 40:773–777.

International Virtual Observatory Alliance (2012). http://www.ivoa.net.

Itoh, M., Akutsu, T., and Kanehisa, M. (2004). Clustering of database sequences for fast homology search using upper bounds on alignment score. *Genome Informatics*, pages 93–104.

James, C. A., Wininger, D., and Delany, J. (2000). Daylight theory manual - daylight 4.71. Technical report, Daylight Chemical Information Systems, Inc.

Kaski, S. (1997). *Data exploration using self-organizing maps*. PhD thesis, Helsinki: Acta Polytechnica Scandinavica.

Kazius, J., McGuire, R., and Bursi, R. (2005). Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medical Chemistry*, 48:312–320.

Khan, D. (2008). Cake - classifying, associating and knowledge discovery - an approach for distributed data mining (ddm) using parallel data mining agents (padmas). In *Web Intelligence and Intelligent Agent Technology. IEEE/WIC/ACM International Conference*, pages 569–601.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, page 671–680.

Kohonen, T. (2001). *Self-organizing maps. 3rd Edition*. Springer.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society*, number 7, pages 48–50.

Larose, D. T. (2005). *Discovering knowledge in data*. Wiley Sons.

Li, W., Jaroszewski, L., and Godzik, A. (2002). Sequence clustering strategies improve remote homology recognitions while reducing search times. *Protein Engineering*, pages 643–649.

Li, Y., Mascagni, M., and Gorin, A. (2009). A decentralized parallel implementation for parallel tempering algorithm. *Parallel Computing*, 35:269–283.

Liao, Q., Yao, J., and Yuan, S. (2007). Prediction of mutagenic toxicity by combination of recursive partitioning and support vector machines. *Molecular Diversity*, 11:59–72.

Lipkus, A. H. (1999). A proof of the triangle inequality for the tanimoto distance. *Journal of Mathematical Chemistry*, 26:263–265.

Liu, X. and Croft, B. W. (2004). Cluster-based retrieval using language models. In *Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval*, pages 186–193. ACM.

Liu, X. and Croft, B. W. (2006). Representing clusters for retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval*, pages 671–672. ACM.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Manning, C. D., Prabhakar, R., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Marinari, E. and Parisi, G. (1992). Simulated tempering: a new monte carlo scheme. *Europhysics Letters*, 19:451–458.

Martinetz, T. M. (1993). Competitive hebbian learning rule forms perfectly topology preserving maps. In *International Conference on Artificial Neural Networks (ICAAN '93)*, pages 427–434.

Martinetz, T. M. and Shulten, K. J. (1991). A neural-gas network learns topologies. *Artificial Neural Networks*, pages 397–402.

Micó, L. and Ocina, J. V. E. (1994). A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15:9–17.

Munk Jørgensen, A., Langgård, M., Gundertofte, K., and Pedersen, J. (2005). A fragment-weighted key-based similarity measure for use in structural clustering and virtual screening. *QSAR Combinatorial Science*, 25(3):221–234.

NASA (2012). http://www.nasa.gov.

National Institute of Standards and Technology (2012). http://webbook.nist.gov/chemistry.

Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, pages 443–453.

Pai, P. S., Nagabhushana, T. N., and Rao, R. B. K. (2007). Growing cell structures (gcs) - a constructive learning neural network for tool wear estimation application. In *International Conference on Conference on Computational Intelligence and Multimedia Applications*, pages 302–319.

Pakhira, M. K. (2008). Fast image segmentation using modified clara algorithm. In *International Conference on Information Technology, ICIT '08*, pages 14–18.

Palma, J. M., Laginha, M., Patrick, R., Amestoy, M. D., Mattoso, M., and Lopes, J. C. (2008). High performance computing for computational science. In *Lectures in Computer Science. VECPAR 8th International Conference*. Springer-Verlag.

Pekalska, E., Duin, R., and Paclik, P. (2006). Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, pages 189–208.

Powell, G. (2005). *Beginning Database Design*. Wiley.

Qiang, X., Cheng, G., and Li, Z. (2010a). A survey of some classic self-organizing maps with incremental learning. In *2nd International Conference on Signal Processing Systems (ICSPS)*, pages 804–809.

Qiang, X., Cheng, G., and Wang, Z. (2010b). An overview of some classical growing neural networks and new developments. In *International Conference on Education Technology and Computer (ICETC)*, pages 351–355.

Rarey, M. and Stahl, M. (2001). Similarity searching in large combinatorial chemistry spaces. *Journal of Computer-Aided Molecular Design*, 15:497–520.

Rathore, N., Chopra, M., and de Pablo, J. J. (2005). Optimal allocation of replicas in parallel tempering simulations. *The Journal of chemical physics*, 122(2):024111.

Rauber, A., Merkl, D., and Dittenbach, M. (2002). The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, pages 1331–1341.

Reforgiato, D. (2009). Grepvs - a combined approach for graph matching. *Journal of Pattern Recognition Research*, pages 14–31.

Riesen, K. (2009). *Classification and Clustering of Vector Space Embedded Graphs*. PhD thesis, Bern University.

Riesen, K., Neuhaus, M., and Bunke, H. (2007). Graph embedding in vector spaces by means of prototype selection. *Graph-Based Representation in Pattern Recognition, GbPRP*, pages 383–393.

Salton, G. (1991). Developments in automatic text retrieval. *Science*.

Sankoff, D. and Kruskal, J. (1999). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison*. CSLI publications.

Satizábal, H. F., Pérez-Uribe, A., and Tomassini, M. (2008). Prototype proliferation in the growing neural gas algorithm. *ICANN*, pages 793–802.

Saux, B. L. and Bunke, H. (2005). Feature selection for graph-based image classifiers. In *Proceedings of 2nd Iberian Conference on Pattern Recognition and Image Analysis LNCS*. Springer.

SETI Institute (2012). http://www.seti.org.

Small, R. D. and Edelstein, H. A. (1997). Scalable data mining tools. In *Building, Using and Managing the Data Warehouse*. Prentice Hall.

Smith, T. and Waterman, M. (1980). New stratigraphic correlation techniques. *Journal of Geology*, pages 451–457.

Stahl, M. and Rarey, M. (2001a). Detailed analysis of scoring functions for virtual screening. *Journal of Medicinal Chemistry*, 44(7):1035–1042.

Stahl, M. and Rarey, M. (2001b). Six sets of active compounds. *Journal of Medical Chemistry*, 44:1035–1042.

Sumathi, S. and Sivanandam, S. N. (2011). *Introduction to Data Mining and its Applications*. Springer-Verlag.

Swendsen, R. and Wang, J. S. (1986). Replica monte carlo simulation of spin glasses. *Physical Review Letters*, 57:2607–2609.

Taniar, D. and Rahayu, J. W. (2002). Parallel data mining. In *Data Mining: A Heuristic Approach*, pages 261–290. Idea Group Publishing.

The VLDB Journal (2012). http://www.vldb.org.

Torres, M. G., Batista, B. M., and Pérez, J. A. M. (2007). Classification and clustering of graphs based on dissimilarity space embedding.

Traina, J. C., Traina, A. J. M., Seeger, B., and Faloutsos, C. (2000). Slim-trees: High performance metric trees minimizing overlap between nodes. In *Proceedings of the 7th International Conference on Extending Database Technology EDBT00*, pages 51–56. Springer-Verlag.

Uhlmann, J. K. (1991). Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 4(40):175–179.

Ullman, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM*, pages 31–42.

Uniprot (2012). http://www.uniprot.org.

Verdonk, M. L., Berdini, V., Hartshorn, M. J., Mooij, W. T. M., Murray, C. W., Taylor, R. D., and Watson, P. (2004). Virtual screening using proteinligand docking: avoiding artificial enrichment. *Journal of Chemical Information and Computer Sciences*, 44(3):793–806.

Viera, M. R., Traina, J. C., and Traina, A. J. M. (2004). Dbm-tree: A dynamic metric access method sensitive to local density data. *Brazilian Symposium on Databases*.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM*, pages 168–173.

Wang, J., Hou, T., and Xu, X. (2009). Aqueous solubility prediction based on weighted atom type counts and solvent accessible surface areas. *Journal of Chemical Information and Modeling*, 49(3):571–581.

Wang, J., Krudy, G., Hou, T., Zhang, W., Holland, G., and Xu, X. (2007). Development of reliable aqueous solubility models and their application in druglike analysis. *Journal of Chemical Information and Modeling*, 47(4):1395–1404.

Wang, Y.-X., Wang, Z.-H., and Li, X.-M. (2004). A parallel clustering algorithm for categorical data set. In *Lecture Notes in Computer Science (Springer)*, pages 928–933.

Weininger, D. (1990). Smiles. 3. depict. graphical depiction of chemical structures. *Journal of Chemical Information and Modeling*, 30:237–243.

Willett, P., Barnard, J. M., and Downs, G. M. (1998). Chemical similarity searching. *Journal of Chemical Information and Computer Sciences*, 38(6):983–996.

Wilson, R. C., Hancok, E. R., and Luo., B. (2003). Pattern vectors from algebraic graph theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 530–549.

Workshop on Algorithms for Modern Massive Data Sets (2010). http://www.stanford.edu/group/mmds.

Xu, R. and Wunsh, D. C. (2010). *Clustering*. Wiley.

Xue, L., Godden, J. W., Stahura, F. L., and Bajorath, J. (2003). Profile scaling increases the similarity search performance of molecular fingerprints containing numerical descriptors and structural keys. *Journal of Chemical Information and Computer Sciences*, 43(4):1218–1225.

Zhang, Y., Xiong, Z., Mao, J., and Ou, L. (2006). The study of parallel k-means algorithm. In *The Sixth World Congress on Intelligent Control and Automation*, pages 5868–5871.

Zhou, D. H. and Bin, L. Y. (2010). An improved birch clustering algorithm and application in thermal power. In *International Conference on Web Information Systems and Mining WISM '10*, pages 53–56.

Zinc Database (2012). http://zinc.docking.org.