UNIVERSIDAD DEL PAIS VASCO
EUSKAL HERRIKO UNIBERTSITATEA
Departamento de Ciencias de la Computación
e Inteligencia Artificial

Algoritmos de Estimación de Distribuciones Aplicados a Problemas Combinatorios en Modelos Gráficos Probabilísticos

Memoria de investigación presentada para optar al título de doctor por
Domingo Romero Asturiano

Tesis dirigida por **Pedro Larrañaga Múgica**

Donostia-San Sebastián, marzo de 2007

Agradecimientos

Deseo expresar mi agradecimiento a todas las personas que, en alguna medida, me han prestado su apoyo desinteresado para que este trabajo de investigación viera la luz. En primer lugar, agradezco al doctor Pedro Larrañaga Múgica, director de la tesis, las ingentes cantidades de tiempo dedicadas a la dirección de esta memoria, así como todos los consejos para la fase de experimentación, sin los cuales este trabajo no habría llegado a buen término.

Agradezco el apoyo y gran ayuda que me ha prestado Basilio Sierra, del departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad del País Vasco, por sus importantes aportaciones para el capítulo sobre aprendizaje estructural. De igual manera, agradezco a Jose Antonio Gámez, del departamento de Informática de la Escuela Politécnica Superior de Albacete, Universidad de Castilla-La Mancha, su ayuda para confeccionar el capítulo sobre inferencia abductiva.

También quiero agradecer las contribuciones de todos los compañeros del departamento de Ciencias de la Computación e Inteligencia Artificial de la UPV/EHU y del Intelligent Systems Group en general, y las de Endika Bengoetxea, Iñaki Inza, Alex Mendiburu, Teresa Miquélez, Roberto Santana y Jose Antonio Lozano en particular.

Igualmente quiero dar las gracias a todos los compañeros del Donostia International Physics Center, centro de investigación en el que trabajo como responsable del Centro de Cálculo, por su apoyo en general, y especialmente por sus aportaciones en la fase de experimentación de la tesis.

Finalmente quiero expresar un agradecimiento especial a mis padres y hermanos por su apoyo constante, sin el cual no habría podido compaginar la confección de esta tesis y mi trabajo, a veces necesariamente absorvente, en el Donostia International Physics Center.

Contenidos

Índice de figuras

Índice de cuadros

Capítulo 1

Presentación

'Cuando las leyes de la matemática se refieren a la realidad, no son ciertas; cuando son ciertas, no se refieren a la realidad.'

Albert Einstein

La Inteligencia Artificial (IA) lleva intentando desde hace muchos años conseguir un sistema capaz de enfrentarse a un problema concreto y solucionarlo tal y como lo haría un ser humano. Esto se ha conseguido en múltiples casos, sobre todo allí donde el conocimiento que maneja el sistema está libre de ambigüedades y es relativamente completo. Sin embargo, en determinados entornos, estos sistemas no se comportan bien debido a la incertidumbre en los datos que manejan, y esta situación se manifiesta sobre todo cuando tratamos con procedimientos generales no adscritos a un problema concreto.

A principios de la década de los ochenta, las técnicas de IA quedaron un tanto desacreditadas al no poder producir los *ordenadores pensantes* que se propusieron crear, aunque en la última decada hay un resurgir de estas técnicas, en particular en lo que se refiere a las relacionadas con la computación evolutiva. El problema principal surge al intentar *imitar* el modo en que el hombre se enfrenta a los problemas, la facilidad para emitir una respuesta ante un determinado patrón. Los procedimientos estocásticos de la IA parecen adaptarse mucho mejor que el resto a estos problemas. Sin embargo, muchos de estos procedimientos carecen de una base sólida, de una teoría que fundamente una generalidad de aplicación, limitando ésta, como siempre, a un problema muy concreto.

La Computación Evolutiva [1] es una de las técnicas heurísticas que más se ha

[1] En (92) puede encontrarse una buena introducción a todo este tipo de técnicas.

desarrollado en las últimas décadas y que más éxito ha tenido en los problemas que ha intentado solucionar. Los algoritmos evolutivos se inspiran en la evolución de las especies, expuesta por primera vez por Darwin (50) y que implica una selección de los mejores individuos y también la existencia de cambios aleatorios.

Computacionalmente hablando, la evolución se puede *simular*. Así, podemos tener comunidades de seres virtuales y estudiar su evolución de manera mas acelerada que en la realidad. Pero también se puede *imitar* para resolver algún problema que en principio no tiene porqué tener relación con lo que se ha dado en llamar *Vida Artificial*. En este último caso, podemos pensar en la Evolución como en una inmensa máquina o sistema que trata de conseguir la optimización de algo. En el caso de la vida real, el *algo* es la adecuación de los seres vivos al contexto natural o psicológico que les rodea. Pero podemos hablar de la evolución de manera más abstracta y no limitar el *algo* a su instanciación en nuestro planeta.

Normalmente, esta *imitación* se traduce computacionalmente en la evolución de determinadas estructuras de datos con el fin de optimizar cierta función matemática, método que se ha venido usando desde hace un par de décadas. El motivo: los métodos clásicos de optimización no pueden aplicarse muchas veces a problemas del mundo real debido a la falta de recursos computacionales lo suficientemente potentes. En cambio, la computación evolutiva sí puede afrontarlos con éxito, a costa de no garantizar la obtención de la solución óptima global. Típicamente, estos algoritmos modifican la estructura provocando *mutaciones* aleatorias en la misma y combinando de alguna manera partes concretas de la misma que se estima tienen una influencia positiva en la optimización de la función. Es decir, que también hay una selección en este caso, aunque no natural, como ocurre en la teoría de Darwin.

De todas estas técnicas, que incluyen a los Algoritmos Genéticos, la Programación Evolutiva, la Programación Genética y las Estrategias Evolutivas, los *Algoritmos de Estimación de Distribución* (EDAs) se han hecho un hueco importante debido a que uno de sus pilares más importantes es la teoria de la probabilidad, lo cual les da una robustez que no tienen otras técnicas basadas mayormente en heurísticos con un componente aleatorio muy determinante.

1.1 Objetivos de la tesis

Este trabajo intenta analizar empíricamente el comportamiento de los algoritmos de estimación de distribuciónes en problemas combinatorios en los que el espacio de búsqueda es considerablemente grande. En concreto, trataremos problemas relacio-

nados con las redes Bayesianas [2], un tipo de modelo gráfico probabilístico del que surgen múltiples problemas de tipo combinatorio. En algunos casos, podremos comparar los resultados proporcionados por los EDAs con otras técnicas evolutivas y no evolutivas. Además, se propone una modificación de los EDAs que permite paliar dos de sus defectos, a saber, el mal comportamiento con problemas en los que el espacio de búsqueda es extremadamente grande, y la ineficiencia con tamaños de individuo grandes.

1.2 Estructura de la tesis

Nos ha parecido adecuado dividir la tesis en un capítulo de presentación, y siete capítulos adicionales. En el capítulo 2 estableceremos la terminología que se utilizará en el resto de la memoria y se presentarán los *modelos gráficos probabilísticos*. En el capítulo 3 se presentarán los *algoritmos de estimación de distribuciones* (EDAs), que son la base de este trabajo. En los siguientes tres capítulos aplicaremos los EDAs a diversos problemas computacionales: en el capítulo 4 estudiaremos el problema de la *obtención de órdenes óptimos en la triangulación de redes Bayesianas*, en el capítulo 5 el de la *obtención de órdenes óptimos para el aprendizaje estructural* y en el capítulo 6 el de la *inferencia abductiva parcial en redes Bayesianas*. Tras constatar los problemas encontrados a lo largo de estos análisis, en el capítulo 7 se propondrá y analizará la aplicación de la *recursividad* a los algoritmos que nos ocupan como una forma elegante de intentar paliarlos. También se propondrá, en el mismo capítulo, una posible paralelización de la estrategia recursiva que incrementaría aún más la eficiencia. Finalmente, en el capítulo 8 intentaremos extraer las conclusiones más generales a las que se ha llegado en los resultados de los capítulos precedentes y se propondrán los trabajos que se pueden llevar a cabo en el futuro para tratar de confirmarlas o quizás de llevarlas aún más lejos.

[2] En (39; 80; 117) pueden encontrarse introducciones sobre los modelos gráficos probabilísticos en general y las redes Bayesianas en particular.

Capítulo 2

Introducción a los Modelos Gráficos Probabilísticos

'La matemática es la ciencia del orden y la medida, de bellas cadenas de razonamientos, todos sencillos y fáciles.'

René Descartes

En este capítulo se introducirán brevemente algunos aspectos de la teoría de la probabilidad y de la teoría de grafos para poder afrontar con algún detalle la teoría de los modelos gráficos probabilísticos (MGP). A continuación ya estaremos preparados para hablar de los algoritmos EDA en el siguiente capítulo, y describir así aquellos que van a ser utilizados en esta memoria. El paradigma de los MGP puede verse con más detalle en (39; 78; 95; 117).

2.1 Medidas de probabilidad

Para llegar a una definición de modelo gráfico probabilístico, antes hemos de realizar otras relacionadas con medidas de probabilidad y la teoría de grafos [1]. A continuación podremos hablar de los modelos probabilísticos propiamente dichos y examinar sus características.

La teoría de la probabilidad nos va a proporcionar una base formal, a la hora de tratar la incertidumbre del mundo real, en la que basar todas las definiciones que seguirán hasta llegar a los algortimos EDA. Llamaremos *Medida de Probabilidad* a

[1] Para una descripción más formal, puede consultarse (87).

una función p que proyecta los subconjuntos $A \subseteq S$ en el intervalo $[0, 1]$ satisfaciendo los siguientes axiomas:

- Normalización. $p(S) = 1$.
- Aditividad. Para cualquier sucesión infinita numerable $A_1, A_2, \ldots$ de subconjuntos disjuntos de S, se cumple que $p(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} p(A_i)$.

Las medidas de probabilidad cumplen las siguientes propiedades:

- Normalización. $p(\emptyset) = 0$.
- Monotonicidad. $A \subseteq B \subseteq S \Rightarrow p(A) \leq p(B)$.
- Continuidad-consistencia. Para toda sucesión creciente $A_1 \subseteq A_2 \subseteq$ o decreciente $A_1 \supseteq A_2 \supseteq$ de subconjuntos de S tenemos que $lim_{i\to\infty} p(A_i) = p(lim_{i\to\infty} A_i)$.
- Inclusión-Exclusión. Dado cualquier par de subconjuntos A y B de S se cumple que $p(A \cup B) = p(A) + p(B) - p(A \cap B)$.

2.1.1 Variable aleatoria y función de densidad

Una variable aleatoria unidimensional X_i es un determinado concepto (una característica variable de un objeto) que puede tener en un momento determinado un valor concreto x_i o *instanciación*. Llamaremos a $p(X_i = x_i)$ la *función de densidad de probabilidad generalizada* en el punto x_i (que abreviaremos como $p(x_i)$). Análogamente, podemos definir una variable aleatoria n-dimensional $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ y una instanciación suya $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ y preguntarnos por la función de densidad $p(\mathbf{x})$, que pasará a llamarse *función de densidad de probabilidad generalizada conjunta*.

2.1.2 Probabilidad marginal

Siendo $p(x_1, \ldots, x_n)$ la probabilidad de que $X_1 = x_1, \ldots, X_n = x_n$ (la función de probabilidad conjunta), entonces la *función de probabilidad marginal* de la i-ésima variable es:

$$p(x_i) = p(X_i = x_i) = \sum_{x_1,,x_{i-1},x_{i+1},\ldots,x_n} p(x_1, \ldots, x_n) \tag{2.1}$$

2.1.3 Probabilidad condicional

Sean $\mathbf{A}$ y $\mathbf{B}$ dos conjuntos disjuntos de variables de una variable multidimensional $\mathbf{X}$ tal que $p(\mathbf{b}) > 0$. Entonces la *probabilidad condicionada* o *función de probabilidad condicionada* de $\mathbf{A}$ dado $\mathbf{B} = \mathbf{b}$ es:

$$p(\mathbf{A} = \mathbf{a}|\mathbf{B} = \mathbf{b}) = p(\mathbf{a}|\mathbf{b}) = \frac{p(\mathbf{a}, \mathbf{b})}{p(\mathbf{b})} \tag{2.2}$$

Análogamente, nos podemos preguntar por la función de probabilidad condicionada de una de las variables de $\mathbf{X}$ (X_i) dado el valor de otra de las variables de $\mathbf{X}$ (X_j):

$$p(X_i = x_i|X_j = x_j) = p(x_i|x_j) = \frac{p(x_i, x_j)}{p(x_j)} \tag{2.3}$$

2.1.4 Variables discretas y variables contínuas

Si X_i es una variable discreta, a $p(x_i)$ se le llama *masa de probabilidad* para la variable X_i. Análogamente, $p(\mathbf{x})$ sería la *masa de probabilidad conjunta* para variables n-dimensionales y $p(x_i|x_j)$ la masa de probabilidad condicional de la variable X_i dado $X_j = x_j$.

En el caso de variables contínuas, hablamos de *función de densidad* $f(x_i)$ de la variable X_i, de función de densidad conjunta $f(\mathbf{x})$ (si todas las componentes o marginales de la variable n-dimensional son contínuas) y, bajo la misma condición, de la función de densidad condicional $f(x_i|x_j)$ de la variable X_i dado $X_j = x_j$.

Podemos generalizar todo esto y hablar de:

- Distribución de probabilidad generalizada $\rho(x_i)$, que en el caso discreto será la masa de probabilidad y en el continuo la función de densidad.
- Distribución de probabilidad generalizada conjunta $\rho(\mathbf{x})$, que en el caso discreto es la masa de probabilidad conjunta y en el continuo la función de densidad conjunta.
- Distribución de probabilidad condicional generalizada $\rho(x_i|x_j)$, que en el caso discreto es la masa de probabilidad condicional y en el continuo la función de densidad condicional.

2.1.5 Independencias

Sean **A** y **B** dos subconjuntos disjuntos de un conjunto de variables aleatorias $\mathbf{X} = (X_1, \ldots, X_n)$. Se dice que **A** es *independiente* de **B** si $p(\mathbf{a}|\mathbf{b}) = p(\mathbf{a})$ para todos los valores posibles **a** y **b** de **A** y **B**. En caso contrario se dice que **A** es *dependiente* de **B**. Un subconjunto de variables aleatorias $\mathbf{A} = (X_1, \ldots, X_m)$ se dice que es independiente cuando $p(x_1, \ldots, x_m) = \prod_{i=1\ldots m} p(x_i)$ para todos los valores posibles $x_1, \ldots, x_m$ de $X_1, \ldots, X_m$. Si las mencionadas variables son condicionalmente independientes con respecto a otro subconjunto $Y_1, \ldots, Y_l$, entonces tenemos que $p(x_1, \ldots, x_m|y_1, \ldots, y_l) = \prod_{i=1\ldots m} p(x_i|y_1, \ldots, y_l)$.

Sean A, B y C tres subconjuntos disjuntos de variables. Se dice que A es *condicionalmente independiente* de B dado C si y sólo si $p(a|c, b) = p(a|c)$ para todos los valores posibles a, b, c de A, B, C. En caso contrario, se dice que A es *condicionalmente dependiente* de B dado C. Una definición equivalente es $p(a, b|c) = p(a|c)p(b|c)$. La relación de independencia condicional se escribe $I(A, B|C)$ y la relación de dependencia condicional, $D(A, B|C)$.

El concepto de independencia/dependencia condicional va a ser clave para la introducción de la definición de red Bayesiana que realizaremos más adelante.

2.1.6 Teorema de Bayes

Denotando por s_i a los síntomas y e_i a las enfermedades, el *teorema de Bayes* se expresa de la siguiente manera:

$$p(e_i|s_1, \ldots, s_n) = \frac{p(e_i, s_1, \ldots, s_n)}{\sum_{e_i} p(e_i, s_1, \ldots, s_n)} = \frac{p(e_i)p(s_1, \ldots, s_n|e_i)}{\sum_{e_i} p(e_i)p(s_1, \ldots, s_n|e_i)} \tag{2.4}$$

A $p(e_i)$ se le llama *probabilidad marginal prior, a priori* o *inicial*, ya que puede ser obtenida antes de contar con los $s_1, \ldots, s_n$. A $p(e_i|s_1, \ldots, s_n)$ se le llama *probabilidad posterior, a posteriori* o *condicional* de e_i, ya que se calcula después de conocer los $s_1, \ldots, s_n$. A $p(s_1, \ldots, s_n|e_i)$ se le llama *verosimilitud* de que dado e_i, se den también $s_1, \ldots, s_n$.

2.1.7 Modelos probabilísticos

Llamaremos *modelo* a una descripción matemática de un fenómeno. En esta descripción habremos de simplificar y omitir determinados detalles, los cuáles no habrán de ser determinantes en el comportamiento del fenómeno: en caso contrario el modelo elegido no será válido. Un modelo nunca va a poder representar con todo detalle

un fenómeno, y a menudo la simplificación va a estar encaminada a poder utilizarlo con un fin específico, esto es, para manejar con el modelo un aspecto determinado del fenómeno, aquel que nos interesa. Esto podría parecer una limitación demasiado grande como para permitirnos aplicar el modelo al mundo real, que es nuestro objetivo. Sin embargo, al fin y al cabo, el modo mediante el cual el ser humano conoce el mundo que le rodea ya impone de por sí una limitación: no conocemos en realidad el mundo real, sino el modelo que del mundo real construye nuestra propia razón a partir de la experiencia de nuestros sentidos.

Podemos hablar de modelos *determinísticos* o de modelos *probabilísticos*; en los primeros los procesos que hemos supuesto implicados en el desarrollo del fenómeno (que serían los síntomas de los que hemos hablado en las secciones anteriores) determinan o no la aparición del fenómeno (esto es, de las enfermedades); en los modelos probabilísticos no hay una determinación rigurosa, sino que se describe el fenómeno en términos de comportamiento probabilístico de los procesos. Es decir, no se habla de si las enfermedades están presentes o no cuando se observan determinados síntomas, sino del comportamiento probabilístico entre enfermedades y síntomas. Muy a menudo, los modelos probabilísticos tienen una relación más estrecha, o pueden servir como descripción más ajustada con respecto al mundo real, en el que nos encontramos información difusa, poco clara y, en resumen, relaciones no deterministas entre unos procesos y otros.

2.1.8 Modelos probabilísticos de síntomas dependientes

En estos modelos los síntomas son dependientes los unos de los otros, y por lo tanto:

$$p(e_i, s_1, \ldots, s_n) = p(s_1, \ldots, s_n)p(e_i|s_1, \ldots, s_n) \tag{2.5}$$

$$p(e_i|s_1, \ldots, s_n) = \frac{p(e_i, s_1, \ldots, s_n)}{p(s_1, \ldots, s_n)} = \frac{p(e_i)p(s_1, \ldots, s_n|e_i)}{p(s_1, \ldots, s_n)} \propto p(e_i)p(s_1, \ldots, s_n|e_i) \tag{2.6}$$

Este es el modelo más realista, pero implica almacenar la función de probabilidad conjunta para todas las combinaciones posibles de los valores de las variables, por lo que normalmente se suelen usar modelos más sencillos como los que veremos a continuación.

2.1.9 Modelos probabilísticos de síntomas independientes

Aquí los síntomas son independientes entre si:

$$p(s_1, \ldots, s_n|e_i) = \prod_{j=1}^{n} p(s_j|e_i) \tag{2.7}$$

$$p(e_i|s_1, \ldots, s_n) = \frac{p(e_i)p(s_1, \ldots, s_n|e_i)}{p(s_1, \ldots, s_n)} = \frac{p(e_i)\prod_{j=1}^{n} p(s_j|e_i)}{p(s_1, \ldots, s_n)} \propto p(e_i)\prod_{j=1}^{n} p(s_j|e_i) \tag{2.8}$$

Como vemos, en este caso la cantidad de información que hemos de almacenar se reduce considerablemente (se limita a las probabilidades marginales $p(e_i)$ y a las probabilidades condicionadas de todos los síntomas con respecto a todas las enfermedades.

2.1.10 Modelos probabilísticos de síntomas relevantes independientes

Si consideramos que las enfermedades dependen siempre de algunos síntomas (llamados *relevantes*, pero no de todos, entonces podemos disminuir aún más la información a manejar. Supongamos que, para la enfermedad e_i, los síntomas relevantes son $s_{r_1}, \ldots, s_{r_i}$ (por tanto, los síntomas $s_{r_i+1}, \ldots, s_n$ son los llamados *irrelevantes*). Supongamos también que para los síntomas irrelevantes s_j la probabilidad condicionada con respecto a las enfermedades e_i para la cuales lo son viene dado por $p_j = p(s_j|e_i)$, siendo idéntica para todos los síntomas. Entonces tenemos que:

$$p(e_i|s_1, \ldots, s_n) = \frac{p(e_i)p(s_1, \ldots, s_n|e_i)}{p(s_1, \ldots, s_n)} = \frac{p(e_i)\prod_{j=1}^{r_i} p(s_j|e_i)\prod_{j=r_i+1}^{n} p_j}{p(s_1, \ldots, s_n)} \propto$$
$$\propto p(e_i)\prod_{j=1}^{n} p(s_j|e_i)\prod_{j=r_i+1}^{n} p_j \tag{2.9}$$

2.1.11 Modelos probabilísticos de síntomas relevantes dependientes

En estos modelos algunos de los síntomas relevantes dependen de otros, lo cual es más realista. En este caso tenemos que:

$$p(e_i|s_1,\ldots,s_n) = \frac{p(e_i)p(s_1,\ldots,s_{r_i}|e_i)\prod_{j=r_i+1}^{n} p(s_j|e_i)}{p(s_1,\ldots,s_n)} =$$
$$= \frac{p(e_i)p(s_1,\ldots,s_{r_i}|e_i)\prod_{j=r_i+1}^{n} p_j}{p(s_1,\ldots,s_n)} \propto p(e_i)p(s_1,\ldots,s_{r_i}|e_i)\prod_{j=r_i+1}^{n} p_j \qquad (2.10)$$

2.2 Grafos

Un grafo o red es un par de conjuntos $G = (X, L)$ donde $X = \{X_1, \ldots, X_n\}$ es un conjunto finito de nodos y L un conjunto de aristas (pares ordenados de elementos distintos de X). Llamemos a la arista que une los nodos X_i y X_j, L_{ij}.

Veamos ahora algunas definiciones posibles sobre la estructura general de un grafo que nos servirán para hablar después de sus propiedades.

2.2.1 Tipos de aristas

Una arista L_{ij} es *dirigida* ($X_i \rightarrow X_j$) si $L_{ij} \in L$ pero $L_{ji} \notin L$, y es *no dirigida* ($X_i - X_j$) en caso contrario. El grafo cuyas aristas son todas dirigidas se llama grafo dirigido y al que tiene a todas sus aristas no dirigidas se le llama grafo no dirigido. En principio, nada impide que dos nodos estén unidos por más de una arista, o que haya una arista que una un nodo consigo mismo. Sin embargo, en las redes Bayesianas y otros tipos de modelos gráficos probabilísticos usaremos los grafos para representar un conjunto de variables proposicionales (que serán los nodos) y unas relaciones de dependencia entre ellas (las aristas), por lo que no será necesario unir dos nodos con más de una arista o que un nodo esté conectado consigo mismo.

2.2.2 Conjunto adyacente

Se llama *conjunto adyacente* del nodo X_i a $\{X_j \in X | L_{ij} \in L\}$ y a partir de ahora lo denotaremos como $Ady(X_i)$. Al conjunto adyacente de un nodo determinado se le llama *vecinos del nodo* si el grafo es no dirigido. Vemos que podríamos representar el grafo como (X, Ady), donde $Ady = \{Ady(X_1), Ady(X_2), \ldots Ady(X_n)\}$. Esta representación va a tener bastantes ventajas desde el punto de vista computacional.

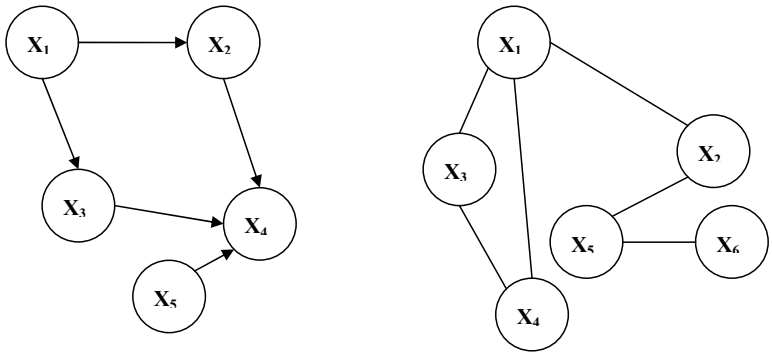

Figura 2.1: Ejemplos de grafo dirigido y no dirigido.

2.2.3 Frontera de un conjunto de nodos

Se llama frontera de un conjunto de nodos S a la unión de los vecinos de los nodos de S excluyendo a los propios nodos de S. La denotaremos como $Frn(S)$:

$$Frn(S) = (\bigcup_{X_i \in S} Ady(X_i)) \setminus S \tag{2.11}$$

2.2.4 Camino entre dos nodos

Un *camino* del nodo X_i al nodo X_j es una sucesión de nodos en la que el primero es X_i, el último X_j y para todos los nodos intermedios existe una arista que va de uno a otro, esto es, $X_{k+1} \in Ady(X_k)$. La longitud del camino es el número de aristas que contiene. Si el primero y el último nodo del camino son el mismo nodo, se dice que el camino es *cerrado*. Un *bucle* es un camino cerrado en un grafo no dirigido. En la Figura 2.1 podemos ver que hay un bucle entre los nodos X_1, X_3 y X_4 en el grafo no dirigido, y podríamos crear un camino cerrado en el dirigido con tal de añadir una arista que fuera del nodo X_4 al X_1. Los caminos cerrados en los grafos dirigidos se llaman *ciclos*.

2.2.5 Grafo completo

Un grafo no dirigido es *completo* si tiene una arista entre cada par de nodos. Un subconjunto S de nodos de un grafo G es completo si existe una arista en G para cada par de nodos de S. En la Figura 2.2 podemos ver a la izquierda un grafo completo. Es evidente que cualquier par de nodos adyacente forma un subconjunto completo.

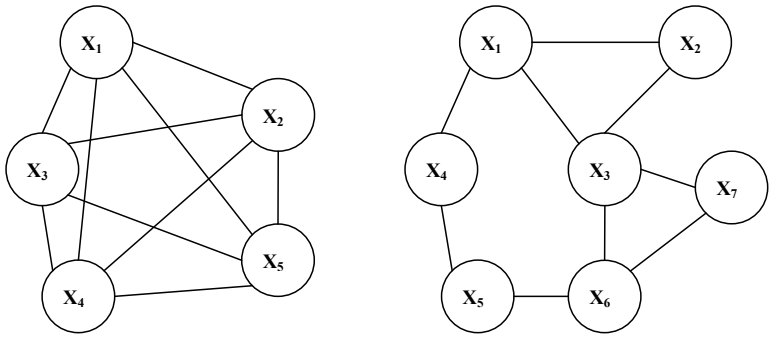

Figura 2.2: Ejemplos de un grafo completo (izquierda) y un grafo no completo (derecha).

2.2.6 Conglomerado o clique

Si el subconjunto S es completo y no es subconjunto de otro conjunto completo (es maximal) entonces se le llama *conglomerado* o *clique*. En la Figura 2.2 podemos ver a la derecha un grafo no completo con los siguientes conglomerados: $C_1 = \{X_1, X_2, X_3\}, C_2 = \{X_1, X_4\}, C_3 = \{X_3, X_6, X_7\}, C_4 = \{X_4, X_5\}$ y $C_5 = \{X_5, X_6\}$.

2.2.7 Grafos conexos y árboles

Un grafo no dirigido es *conexo* si existe al menos un camino entre cada par de nodos (si no lo hay, se denomina *inconexo*). Si el camino es siempre único, entonces se llama *árbol*. Si al menos dos nodos están unidos por más de un camino (hay por lo menos un bucle), entonces se habla de *grafo múltiplemente conexo*. Obsérvese que en un árbol la eliminación de cualquier arista lo divide en dos subgrafos inconexos.

2.2.8 Parentescos de los nodos

Si existe la arista $X_i \rightarrow X_j$, se dice que X_i es un *padre* de X_j, y que X_j es un hijo de X_i. Al conjunto de padres del nodo X_i lo denotaremos mediante Π_i, y sus valores se denotarán por medio de π_i. El conjunto de un nodo y sus padres es la *familia* del nodo. X_j es *ascendiente* de X_i si existe un camino de X_j a X_i (X_i sería *descendiente* de X_j). Un conjunto de nodos se denomina *conjunto ancestral* de otro conjunto de nodos dado si contiene los ascendientes de todos los nodos de este segundo conjunto.

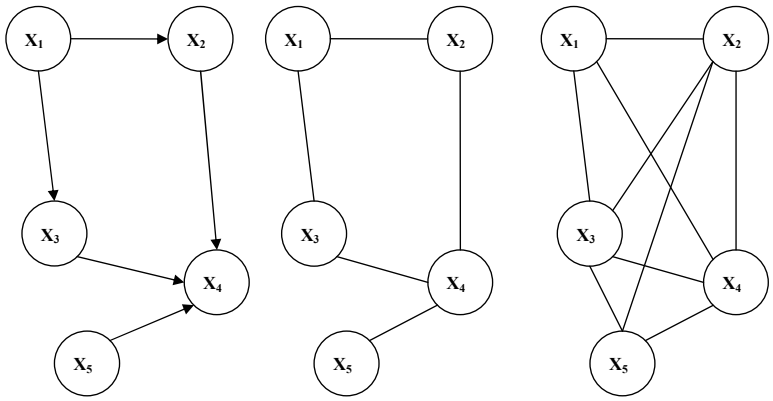

Figura 2.3: Proceso que lleva de un grafo dirigido a uno moral.

2.2.9 Ordenación y numeración ancestral

Dado un conjunto $X = \{X_1, X_2, \ldots, X_n\}$ de nodos, llamamos *ordenación* de X a una biyección α entre el conjunto $\{1, 2, \ldots, n\}$ y X:

$$\alpha : \{1, 2, \ldots, n\} \rightarrow \{X_1, X_2, \ldots, X_n\} \tag{2.12}$$

El índice en el orden α que tiene un nodo X_i lo denotaremos como $\alpha^{-1}(X_i)$, y el nodo que ocupa el índice σ en el orden α lo denotaremos como $\alpha(\sigma)$. Así, si decimos que dados un X_i y un X_j, $i <_\alpha j$, nos referimos a que X_i es anterior a X_j en el orden α. Una numeración de los nodos de un grafo dirigido es *ancestral* si el número correspondiente a cada nodo es menor que los correspondientes a sus hijos.

2.2.10 Grafo moral

Dado un grafo dirigido, si sustituimos sus aristas por aristas no dirigidas obtendremos el grafo no dirigido asociado. Si añadimos una arista a cada par de nodos que tienen un hijo en común en el grafo no dirigido, obtenemos el *grafo moral*. En la Figura 2.3 podemos ver el proceso completo que lleva de un grafo dirigido a uno moral.

2.2.11 Equivalencias entre grafos dirigidos y no dirigidos

Un grafo dirigido es *conexo* si el no dirigido asociado lo es (e inconexo en caso contrario). Si el no dirigido es un árbol, el dirigido también se llamará *árbol* (o *grafo*

múltiplemente conexo en caso contrario).

2.2.12 Grafos cíclicos

Si el grafo dirigido contiene por lo menos un ciclo, se le llama *grafo cíclico*. En caso contrario tendremos un grafo dirigido *acíclico*. Éstos últimos van a ser la base del modelo gráfico probabilístico que vamos a usar en el presente trabajo, las *redes Bayesianas*.

2.2.13 Poliárboles

Si los nodos de un árbol dirigido tienen como máximo un padre, entonces tenemos un *árbol simple*. En caso contrario tenemos un *poliárbol*.

2.2.14 Grafo triangulado

Se llama *cuerda* a una arista que une dos nodos de un bucle pero que no pertenece al bucle. Un *grafo triangulado* o *cordal* tiene, en cada bucle de longitud mayor o igual que cuatro, una cuerda. A este tipo de grafos también se les llama *circuitos rígidos*, y constituyen la estructura gráfica de los MGP llamados *modelos descomponibles* (96), con gran variedad de aplicaciones prácticas. En la Figura 2.4 podemos ver un grafo que no es triangulado y uno equivalente que sí lo es. Para convertir un grafo no triangulado en uno que lo sea, basta con añadir cuerdas que dividan los bucles, proceso que se llama *triangulación*. Se habla de triangulación *minimal* cuando añadimos el mínimo número de aristas que son necesarias para triangular el grafo, que es lo que nos va a interesar para preservar lo máximo posible la topología original del grafo. Hallar la triangulación minimal es un problema NP-duro (156), y volveremos sobre este tema en el capítulo 4.

2.2.15 Grafo de aglomerados

Se llama *aglomerado* a un conjunto de nodos de un grafo. Sea el grafo $G = (X, L)$ y un conjunto de aglomerados $C = \{C_1, C_2 \ldots, C_m\}$ tal que $X = C_1 \cup C_2 \cup \ldots \cup C_m$. El grafo $G' = (C, L')$, donde las aristas de L' unen aglomerados que tienen algún nodo en común, se llama *grafo de aglomerados* de G.

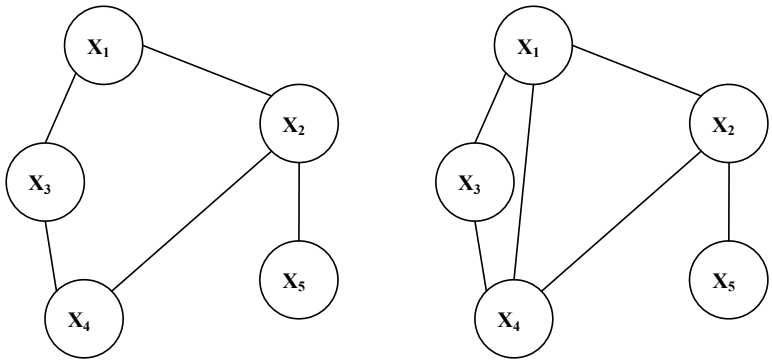

Figura 2.4: Grafo no triangulado (el bucle $X_1 - X_2 - X_4 - X_3$ no contiene ninguna cuerda) y el triangulado equivalente (añadiendo la cuerda $X_1 - X_4$).

2.2.16 Grafo de conglomerados

Un grafo de aglomerados se llama *grafo de conglomerados* asociado a un grafo no dirigido G si sus aglomerados son los conglomerados de G.

2.2.17 Árbol de unión

Un grafo de conglomerados se llama *árbol de unión* si es un árbol y todo nodo que pertenece a dos conglomerados también pertenece a los que se hallan en el camino que los une. Si no es un árbol, hablamos de *grafo de unión*. Se da el hecho de que un grafo no dirigido posee un árbol de unión si y sólo si es triangulado.

2.2.18 U-separación

Sean X, Y y Z tres subconjuntos disjuntos de nodos de un grafo no dirigido G. Se dice que Z *U-separa* X e Y si y solo si cada camino entre nodos de X y nodos de Y tiene algún nodo de Z. Se denota $I(X, Y|Z)_G$ y se dice que X e Y son *condicionalmente independientes* dado Z. Si la condición no se cumple, entonces tenemos que $D(X, Y|Z)_G$ y se dice que X e Y son condicionalmente dependientes dado Z.

2.2.19 Nodo de aristas convergentes

Dado un grafo dirigido y un camino $\ldots - U - A - V - \ldots$, el nodo A se denomina *nodo de aristas convergentes* en el camino si en el grafo dirigido existen las aristas $U \rightarrow A$ y $V \rightarrow A$.

2.2.20 D-separación

Sean X, Y y Z tres subconjuntos disjuntos de nodos de un grafo dirigido acíclico G. Se dice que Z *D-separa* a X e Y si en todos los caminos que van de un nodo de X a un nodo de Y existe un nodo intermedio A que, o bien es de aristas convergentes en el camino (y ni A ni sus descendientes están en Z) o bien A no es de aristas convergentes en el camino pero está en Z. Como antes, la relación de independencia se notará $I(X, Y|Z)_G$ (o $D(X, Y|Z)_G$ en caso de que no se dé, esto es, que haya una relación de dependencia dado Z). Otra definición de la D-separación es la siguiente: Z D-separa a X e Y si y sólo si Z U-separa X e Y en el grafo moral del menor subconjunto ancestral que contenga a los nodos de X, Y y Z.

2.2.21 Lista inicial de independencias

Se llama lista inicial de independencias a un conjunto de relaciones de independencia de la forma $I(X, Y|Z)$, siendo X, Y y Z disjuntos. En una lista de independencias se pueden verificar o no las siguientes propiedades:

- Simetría. $I(X, Y|Z) \iff I(Y, X|Z)$.
- Descomposición. $I(X, Y \cup W|Z) \Rightarrow I(X, Y|Z) \wedge I(X, W|Z)$. La propiedad recíproca se conoce como composición, pero no se cumple en todos los modelos probabilísticos.
- Unión débil. $I(X, Y \cup W|Z) \Rightarrow I(X, W|Z \cup Y) \wedge I(X, Y|Z \cup W)$.
- Contracción. $I(X, W|Z \cup Y) \wedge I(X, Y|Z) \Rightarrow I(X, Y \cup W|Z)$.
- Intersección. $I(X, W|Z \cup Y) \wedge I(X, Y|Z \cup W) \Rightarrow I(X, Y \cup W|Z)$.
- Unión fuerte. $I(X, Y|Z) \Rightarrow I(X, Y|Z \cup W)$.
- Transitividad débil. $D(X, A|Z) \wedge D(A, Y|Z) \Rightarrow D(X, Y|Z) \vee D(X, Y|Z \cup A)$.
- Transitividad fuerte. $D(X, A|Z) \wedge D(A, Y|Z) \Rightarrow D(X, Y\ Z)$.

2.2.22 Grafoides y semigrafoides

Un conjunto de relaciones de independencia se llama *grafoide* si es cerrado con respecto a las propiedades de simetría, descomposición, unión débil, contracción e intersección. Si esta última propiedad no es cerrada, se habla de un *semigrafoide*.

2.2.23 Modelo de dependencia

Cualquier modelo M de un conjunto de variables $\{X_1, X_2, \ldots, X_n\}$ mediante el cual se pueda determinar si $I(X, Y|Z)$ es o no cierta para todas las posibles ternas de subconjuntos X, Y y Z se denomina *modelo de dependencia*. Ejemplos de modelos de dependencia son una lista inicial de independencias, un grafo o una función de probabilidad conjunta.

2.2.24 Mapas de modelos de dependencia

Un grafo G se denomina un *mapa perfecto de un modelo de dependencia* M si $I(X, Y|Z)_M \iff I(X, Y|Z)_G$. Dependiendo del grafo tendremos mapas perfectos dirigidos o no dirigidos. Si $I(X, Y|Z)_G \Rightarrow I(X, Y|Z)_M$ se habla de *mapa de independencia* o *I-mapa* (esto implica que $D(X, Y|Z)_M \Rightarrow D(X, Y|Z)_G$). Si cambiamos independencias por dependencias y dependencias por independencias, tenemos los mapas de dependencia o D-mapas. Si un I-mapa de un modelo de dependencias M pierde esta propiedad cuando se le quita cualquier arista, tenemos un I-mapa *minimal*.

2.2.25 Hipergrafo

En los hipergrafos dirigidos, el conjunto de aristas (aquí llamadas *hiperaristas*) está compuesto por elementos $l = (t(l), h(l))$, donde $t(l), h(l) \in X$ y son tales que $t(l) \cap h(l) = \varnothing$ y $t(l) \cup h(l) \neq \varnothing$.

A $t(l)$ se le llama *cola* de la hiperarista l, y a $h(l)$ *cabeza*. En un hipergrafo el conjunto que nodos que determina una hiperarista tiene, a diferencia de los grafos, una cardinalidad superior a 2. En los hipergrafos no dirigidos no hay distinción entre cabeza y cola.

2.3 Modelos gráficos probabilísticos

Sea $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ un conjunto de variables aleatorias, donde x_i es una instanciación de X_i e $\mathbf{y}$ una instanciación de $\mathbf{Y} \subseteq \mathbf{X}$. Se denomina *modelo gráfico probabilístico* a una factorización gráfica de la distribución de probabilidad generalizada conjunta $\rho(\mathbf{X} = \mathbf{x})$ o $\rho(\mathbf{x})$ realizada mediante una estructura gráfica y un conjunto de distribuciones de probabilidad locales referidas a la estructura. En general, tenemos que un modelo gráfico probabilístico es un par (S, θ_S). S es la estructura gráfica en la que los nodos van a ser variables aleatorias. La probabilidad de una determinada instanciación de las variables aleatorias es:

$$\begin{aligned} \rho(\mathbf{x}) = \rho(x_1, \ldots, x_n) = \\ = \rho(x_1) \cdot \rho(x_2|x_1) \cdot \ldots \cdot \rho(x_i|x_1, \ldots, x_{i-1}) \cdot \ldots \cdot \rho(x_n|x_1, \ldots, x_{n-1}) \end{aligned} \tag{2.13}$$

y en virtud de la factorización que determina la estructura gráfica, la función de densidad nos queda como:

$$\rho(\mathbf{x}) = \prod_{i=1,\ldots,n} \rho(x_i|\pi_i) \tag{2.14}$$

lo que nos permite tener unas distribuciones de probabilidad generalizada locales. Asumimos que éstas dependen de un conjunto finito de parámetros $\theta_{\mathbf{S}} = (\theta_1, \ldots, \theta_n)$, esto es, uno por variable, por lo que la ecuación anterior puede reescribirse como:

$$\rho(\mathbf{x}|\theta_S) = \prod_{i=1,\ldots,n} \rho(x_i|\pi_i, \theta_i) \tag{2.15}$$

La ventaja de los modelos gráficos, o los modelos definidos por listas de independencias a la hora de construir un modelo probabilístico es que definen una factorización de la distribución de probabilidad generalizada conjunta como producto de distribuciones de probabilidad condicional generalizada que nos hablan de la estructura cualitativa del modelo. Las condicionadas van a manejar un número menor de variables que la conjunta, lo que hace más sencilla la definición del modelo probabilístico.

2.3.1 Redes Bayesianas

Una red Bayesiana es un modelo gráfico probabilístico en el que la estructura S es un grafo dirigido acíclico representando los nodos variables aleatorias discretas. Es

decir, la *fuerza* de las suposiciones de independencia entre las diversas variables es expresada en términos de probabilidades condicionales. El conjunto de parámetros θ_S es el conjunto de las n funciones de probabilidad condicionada (una por variable). Se da el hecho de que S es un I-mapa minimal de $p(\mathbf{x})$.

S puede ser, en este caso y en orden de complejidad, un árbol simple (cada variable tiene como mucho un padre), un poliárbol (tiene de 0 a n padres, pero los padres se consideran mútuamente independientes) o un grafo múltiplemente conexo.

Al ser las variables discretas, X_i tiene r_i posibles valores $x_i^1, \ldots, x_i^{r_i}$. Si denotamos como q_i el número de posibles instanciaciones de los padres de X_i, esto es, $q_i = \prod_{X_g \in \Pi_i} r_g$, y como π_i^j a la j-ésima instanciación, tenemos que la distribución local para una variable concreta dada una determinada instanciación de sus padres nos queda como:

$$p(x_i^k | \pi_i^j, \theta_i) = \theta_{ijk} \tag{2.16}$$

En la Figura 2.5 podemos ver un ejemplo de red Bayesiana. Las redes Bayesianas proporcionan una manera más sencilla de construir modelos probabilísticos que otro tipo de redes, como las de Markov, en las que antes hay que construir los conglomerados del grafo, ordenarlo de manera que satisfaga determinadas propiedades, etc. Con las redes Bayesianas sólo necesitamos el grafo acíclico y el conjunto de probabilidades condicionadas.

2.3.2 Modelos gráficos equivalentes

Si los grafos correspondientes a dos modelos gráficos determinan el mismo modelo de dependencia (dan lugar al mismo conjunto de relaciones de independencia) entonces se habla de *modelos gráficos equivalentes*.

2.3.3 Equivalencias de redes Bayesianas

Una terna de nodos (X, Y, Z) en una red Bayesiana es una *v-estructura* si las aristas desde los nodos X e Y convergen en el nodo Z y no hay ninguna arista entre X e Y. A Z se le denomina *nodo de aristas convergentes no acopladas* en el camino no dirigido $X - Z - Y$. Dos redes Bayesianas son *equivalentes* si tienen asociados los mismos grafos no dirigidos y las mismas v-estructuras.

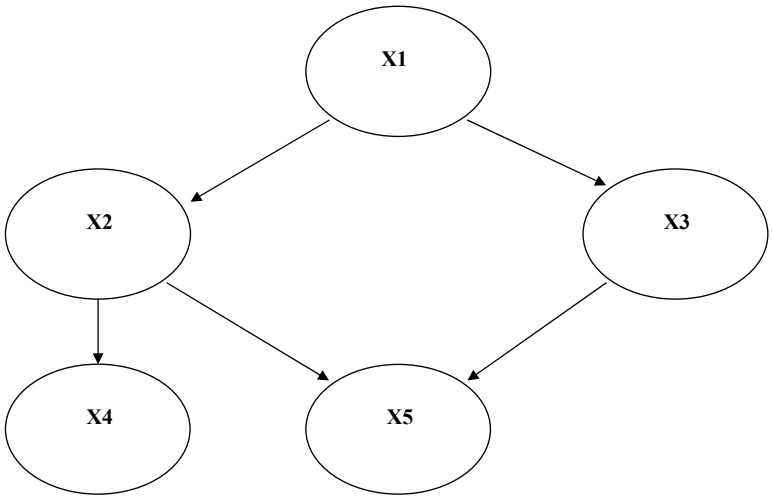

Figura 2.5: Ejemplo de red Bayesiana.

2.3.4 Grados de libertad de una red Bayesiana

Dada la definición de red Bayesiana introducida en las secciones anteriores, vemos que a la hora de construir una red concreta tenemos que determinar lo siguiente:

- El grafo dirigido acíclico que sustenta el conjunto de independencias condicionales entre las variables, problema que se ha dado en llamar *aprendizaje estructural*.
- Las probabilidades a priori de todos los nodos raíz (cuyo conjunto de padres es el vacío).
- El conjunto de las probabilidades condicionales para el resto de los nodos, esto es, los θ_{ijk}, problema que se denomina *aprendizaje paramétrico*.

Llamaremos *grados de libertad* de una red a todos los elementos estructurales y paramétricos que no están prefijados con anterioridad a su construcción.

2.3.5 Redes Gausianas

Una *red Gausiana* (138) es un modelo gráfico probabilístico en el que la estructura S es un grafo dirigido acíclico y las variables aleatorias que conforman los nodos son contínuas. Además, la función de densidad condicional local es el modelo de regresión lineal. Siguiendo la terminología escogida para las redes Bayesianas, tenemos que:

$$f(x_i|\pi_i, \theta_i) = \aleph(x_i; m_i + \sum_{x_j \in \pi_i} b_{ji}(x_j - m_j), v_i) \tag{2.17}$$

donde $\aleph(x; \mu, \sigma^2)$ es una distribución normal univariada con media μ y varianza $\sigma^2 > 0$. Ahora el vector de parámetros consiste en $\theta_i = (m_i, \mathbf{b_i}, v_i)$, donde $\mathbf{b_i} = (b_{1i}, \ldots, b_{i-1i})^t$ es un vector columna tal que si no hay arco de X_j a X_i, entonces $b_{ji} = 0$. m_i es la media incondicional de X_i y v_i es la varianza condicionada de X_i dados sus padres π_i.

Hay una relación estrecha entre las redes Gausianas y las densidades normales multivariantes. Se considera que la función de densidad conjunta de una variable contínua n-dimensional es una distribución normal multivariante si cumple:

$$f(\mathbf{x}) \equiv \aleph(\mathbf{x}; \mu, \Sigma) \equiv (2\pi)^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^t \Sigma^{-1}(\mathbf{x}-\mu)} \tag{2.18}$$

donde μ es el vector de esperanzas matemáticas, Σ es la matriz de covarianzas y $|\Sigma|$ es el determinante de Σ. La inversa de esta última matriz $W = \Sigma^{-1}$ de elementos que notaremos w_{ij} se llama *matriz de precisión*. Se da el caso de que la densidad anterior puede ser escrita como un producto de n densidades condicionales:

$$f(\mathbf{x}) = \prod_{i=1}^{n} f(x_i|x_1, \ldots, x_{i-1}) = \prod_{i=1}^{n} \aleph(x_i; \mu_i + \sum_{j=1}^{i-1} b_{ji}(x_j - \mu_j), v_i) \tag{2.19}$$

μ_i es la media incondicional de X_i y v_i es la varianza de X_i dados $X_1, \ldots, X_{i-1}$. Con esta notación, podemos interpretar una distribución normal multivariada como una red Gausiana en la que b_{ji}, que es un coeficiente lineal que refleja la fuerza de la relación entre el nodo X_j y el nodo X_i, determina la existencia de un arco entre X_j y X_i cuando es distinto de 0 con $j < i$. Además, $m_i \equiv \mu_i$. De igual manera, dada una red Gausiana, podemos generar una densidad normal multivariante, teniendo en cuenta que $\mu_i \equiv m_i$ y que podemos obtener la matriz de precisión W a partir de los v_i y b_i aplicando la siguiente fórmula recursiva:

$$W(i+1) = \begin{cases} \frac{1}{v_1} & \text{si } i = 0; \\ \begin{pmatrix} W(i) + \frac{b_{i+1} b_{i+1}^t}{v_{i+1}} & -\frac{b_{i+1}}{v_{i+1}} \\ -\frac{b_{i+1}^t}{v_{i+1}} & \frac{1}{v_{i+1}} \end{pmatrix} & \text{si } i > 0. \end{cases}$$

Esta relación nos va a ser útil a la hora de realizar un muestreo de la red, como veremos más adelante.

2.3.6 Grados de libertad de una red Gausiana

Dada la definición de red Gausiana, vemos que a la hora de construir una red determinada tenemos que concretar lo siguiente:

- El grafo dirigido acíclico que sustenta el conjunto de independencias condicionales entre las variables.
- La colección de parámetros locales para cada variable, esto es, $\theta_i = (m_i, \mathbf{b_i}, v_i)$.

2.4 Problemas asociados al uso de modelos gráficos probabilísticos

Los grados de libertad asociados a una red implican otros tantos problemas a resolver a la hora de construir un modelo probabilístico. A estos hay que añadir el problema de la propagación de la evidencia, esto es, el cálculo de $\rho(X_i|X_j)$. Otro uso de MGP que conlleva su correspondiente dificultad consiste en su *simulación*. En (75) hay una buena introducción a todo este tipo de cuestiones. A lo largo de este trabajo nos limitaremos al uso de dos tipos de modelos gráficos concretos: las ya mencionadas redes Bayesianas y las redes Gausianas, que son los dos tipos de MGP implicados en el funcionamiento de los EDA que se utilizarán en los experimentos.

2.4.1 Aprendizaje estructural

El capítulo 5 está dedicado al aprendizaje estructural, y en él se trata este tema con algo más de detalle. Se trata de un problema NP-duro si cada nodo puede tener un número de padres mayor que uno, como se demuestra en (42). La estructura de la red puede ser construida por un experto en el problema que intentamos resolver, pero éste método está a menudo sujeto a errores e imprecisiones. A menudo es complicado

formalizar el conocimiento con el que cuenta el experto, que muchas veces puede resolver un problema movido por su intuición, por lo que esta manera de modelar puede llevar a consumir mucho tiempo.

Sin embargo tenemos otra opción. Podemos intentar generar la estructura a partir de un conjunto de datos observados de tamaño N de un dominio $X = (X_1, \ldots, X_n)$ con n variables. Esto tampoco es la panacea, pues hay que seleccionar a priori qué conjunto de variables observables tiene algo que ver en el comportamiento de un fenómeno determinado, pero determinar esto en vez de la estructura completa es un problema algo menos complicado para un experto. ¿Qué estructura es la que refleja más fielmente dicho conjunto de datos? Probar todas las estructuras posibles para comprobar cómo se ajustan a los datos observados no es una buena política, ya que el espacio de las estructuras crece exponencialmente con el número de nodos (126).

Normalmente, ya que la solución exacta es inviable para redes grandes, se realizan restricciones en el tipo de estructuras posibles a fin de reducir el espacio de búsqueda. Por ejemplo, se tienen en cuenta sólo árboles (43) o poliárboles (123). Hay que tener presente que aquí hay dos subproblemas. Por un lado necesitamos un algoritmo que genere estructuras más o menos óptimas, pero también otro algoritmo que, dada una estructura, nos diga lo cercana que está a explicar el conjunto de datos observados, una *función de evaluación* o métrica de la bondad de la red.

En redes múltiplemente conectadas las restricciones han de ser mayores, lo que hace que la estructura resultante pueda llegar a estar muy lejos de lo que buscamos, por lo que se impone una búsqueda heurística utilizando algoritmos aproximados. Podemos partir del árbol Bayesiano con el algoritmo de Chow y Liu (43), de una red construida aleatoriamente o simplemente de una red sin ningún arco, y luego ir añadiendo los arcos que maximicen la función de evaluación. Muchas veces no es preferible una red muy conectada, esto es, con muchos arcos, aunque explique mejor el conjunto de datos, pues un número de arcos grande hace más complejo otro de los problemas al que nos enfrentamos: el de la propagación de la evidencia. En este caso hay que llegar a un compromiso entre fidelidad y eficiencia. En el capítulo 5, basado en el trabajo que aparece en (128), se utiliza para generar la estructura el algoritmo K2 de Cooper y Herskovits (45), que implica un ordenamiento previo $<_o$, de las variables y la restricción $X_i \in \pi_j \Rightarrow i <_o j$. El algoritmo de Chow y Liu también implica un ordenamiento previo de las variables, implícito en el árbol, limitando qué padres puede llegar a tener cada nodo. En (20) se utilizan EDAs (UMDA y PBIL en este caso) para la búsqueda directa en el espacio de estructuras, otra vez restringido el mismo en cuanto al tipo de arcos que se pueden añadir. También se puede usar el enfriamiento estadístico como heurístico de búsqueda. En cualquier

caso, siempre hay un algoritmo que añade arcos y una función que evalúa la bondad de los arcos añadidos.

El problema principal de las búsquedas heurísticas de la estructura es que pueden acabar en un máximo local o en una zona del espacio de búsqueda en la que la dinámica de adición de arcos que lleva a cabo nuestro algoritmo no mejora la función de evaluación (aunque si existan arcos posibles que la mejoren). Hay métodos para *escapar* de este tipo de situaciones, como la búsqueda TABU, el enfriamiento estadístico o un simple reinicio aleatorio del algoritmo. Como hemos comentado antes, volveremos sobre todos estos temas en el capítulo 5.

En lo que se refiere a la función de evaluación, el algoritmo $K2$ por ejemplo lleva asociada ya una concreta, la *métrica de la verosimilitud marginal*, que tiene en cuenta el conjunto de datos y distribuciones a priori sobre las estructuras y sobre los parámetros, aunque pueden utilizarse otras que también evalúen la complejidad de la red resultante, como por ejemplo la métrica de la máxima verosimilitud penalizada, o métricas basadas en la teoría de la información. En el trabajo de Blanco (20) puede verse una comparación de los resultados obtenidos con distintas funciones de evaluación de la red. En nuestro caso, veremos más adelante que algunos EDAs utilizados en los experimentos, internamente, deben *aprender* una red Bayesiana a partir de una población dada, para lo cual utilizarán el algoritmo $K2$, que como ya hemos dicho se comenta con detenimiento en el capítulo 5.

En cuanto a las redes Gausianas, también se han propuestos varios métodos para realizar la inducción del modelo. Podemos modelar la estructura de la matriz $n \times n$ de precisión W investigando si cada elemento w_{ij}, con $i = 1, \ldots, n-1$ y $j > i$, puede ser cero (54). La idea es simplificar así la funcion de densidad normal n-dimensional conjunta. Esto es equivalente a realizar tests de independencia condicional entre los elementos correspondientes de $\mathbf{X}$ (152) o a comprobar si el arco que conecta los vértices X_i y X_j en el grafo de independencia condicional puede ser eliminado (144). De ahí el nombre más conocido de este método, el de *tests de exclusión de arcos*. También hay métodos de búsqueda basados en métricas como los comentados para el caso de variables discretas, los cuales también son válidos para las redes Gausianas, para las que se utilizan, obviamente, métricas diferentes a la hora de examinar la bondad de la red. La métrica de la *verosimilitud* $L(D|S, \theta)$ tiene la expresión:

$$\sum_{r=1}^{N} \sum_{i=1}^{n} \left[-\log(\sqrt{2\pi v_i} - \frac{1}{2v_i} \left(x_{ir} - m_i - \sum_{x_j \in \pi_i} b_{ji}(x_{jr} - m_j) \right)^2 \right] \tag{2.20}$$

donde x_{ir} es la i-ésima componente del r-ésimo elemento del conjunto de datos D. Esta expresión se puede penalizar, hablándose entonces de la *verosimilitud penalizada* $L(D|S,\theta) - f(N)\cdot(2n + \sum_{i=1}^{n} |\pi_i|)$. La función $f(N)$ es una función de penalización no negativa que puede tomar distintos valores según los autores ($f(N) = 1$ para el criterio de información de Akaike (3), $f(N) = \frac{1}{2} log N$ para el criterio de Jeffreys-Schwarz (134) por citar dos ejemplos). Geiger y Heckerman (71) proponen una métrica Bayesiana llamada *equivalencia Bayesiana Gausiana* que tiene la interesante propiedad de asignar el mismo valor a dos redes Gausianas si son isomórficas, esto es, representan las mismas afirmaciones de dependencia e independencia condicional. La métrica está basada en el hecho de que la distribución normal-Wishart es conjugada con la normal multivariante, lo que permite obtener una fórmula para calcular la verosimilitud marginal del fichero de muestras dada una estructura.

2.4.2 Aprendizaje paramétrico

El aprendizaje paramétrico consiste en la identificación de parámetros numéricos del modelo gráfico probabilístico. En el caso de una red Bayesiana, los θ_{ijk}, que no son más que las probabilidades condicionadas, son fácilmente extraíbles de una hipotética base de datos de casos.

En el caso de una red Gaussiana, podemos obtener el estimador máximo verosímil $\widehat{\theta} = (\widehat{\theta_1}, \ldots, \widehat{\theta_n})$ del vector de parámetros de la red maximizando la expresión de la métrica de la verosimilitud ya vista en la sección anterior y solucionando el siguiente sistema de ecuaciones:

$$\begin{cases} \frac{\partial}{\partial m_i} \log L(D|S,\theta) = 0 & i = 1, \ldots, n \\ \frac{\partial}{\partial v_i} \log L(D|S,\theta) = 0 & i = 1, \ldots, n \\ \frac{\partial}{\partial b_{ji}} \log L(D|S,\theta) = 0 & j = 1, \ldots, i-1 \wedge X_j \in \pi_i. \end{cases}$$

Las soluciones a este sistema nos dan expresiones para $\widehat{m_i}$, $\widehat{b_{ji}}$ y $\widehat{v_i}$ basadas en la media muestral de X_i, la covarianza muestral entre las variables X_j y X_i y la varianza muestral de X_j (92).

2.4.3 La propagación de la evidencia

El problema de la propagación de la evidencia (el cálculo de $\rho(X_i|X_j)$ está descrito con detalle para las redes Bayesianas en el capítulo 4, dedicado a la triangulación. Hay varias maneras de propagar la evidencia en una red, y una de las más utilizadas

es el algoritmo de Lauritzen y Spiegelhalter (97), lo cual implica un proceso de eliminación de variables de la red llamado triangulación. Este proceso es especialmente importante, porque determina la velocidad de la propagación de la evidencia. Es decir, una buena triangulación nos permite una rápida propagación.

2.4.4 La simulación de las redes

La simulación será utilizada en este trabajo para generar los individuos de los EDA en cada generación (ver capítulo 3). En el caso de las redes Bayesianas, se parte de una red no instanciada (el valor de las variables del sistema es desconocido) y se utiliza para asignar aleatoriamente un valor a todas las variables. Es evidente que el valor de un nodo padre determina, en función de la probabilidad condicionada, el posible valor que puedan llegar a tener sus hijos. Así, podemos utilizar la red como un generador de escenarios. La simulación es también una alternativa a la propagación exacta de la evidencia, pues con ella podemos generar una base de datos en la que las relaciones probabilísticas entre las variables estén subyacentes.

En los últimos años se han propuesto un buen número de métodos de simulaciones de redes Bayesianas, como el *método de muestreo de Markov* de Pearl (117), el *método de la función de verosimilitud pesante*, desarrollado independientemente por Fung y Chang (66) y por Shachter y Peot (137), el *Muestreo Hacia Adelante y Hacia Atrás* propuesto por Fung y Del Favero (65) o el *Método de Muestreo Sistemático* de Bouckaert (29). El método más utilizado para variables discretas es el *muestreo lógico probabilístico* o PLS, propuesto por Henrion (77). En él, se obtiene una instancia de una variable una vez que todos sus padres han sido instanciados previamente. Para ello las variables se ordenan previamente en orden ancestral (los padres antes que los hijos). Así, usamos un número aleatorio para producir una muestra de cada variable raíz de la red, teniendo en cuenta su probabilidad a priori si la hubiera, y después se simula un valor para cada variable del resto de la red (una a una) teniendo en cuenta las probabilidades condicionadas entre los padres de la variable y la propia variable.

En el caso de las variables contínuas, es decir, en el caso de las redes Gausianas, Ripley propone en (124) dos aproximaciones generales al muestreo de distribuciones normales multivariadas, que como hemos visto antes pueden ser obtenidas a partir de una red Gausiana y viceversa. La primera aproximación se basa en la descomposición de Cholesky de la matriz de covarianzas. La segunda, conocida como el *método de condicionamiento*, genera instancias de $\mathbf{X}$ computando X_1, luego X_2 condicionado a X_1, etc, lo que tiene ciertas similitudes con el algoritmo PLS del caso discreto, y es la opción que se utlizará en nuestros experimentos. Para simular una distribu-

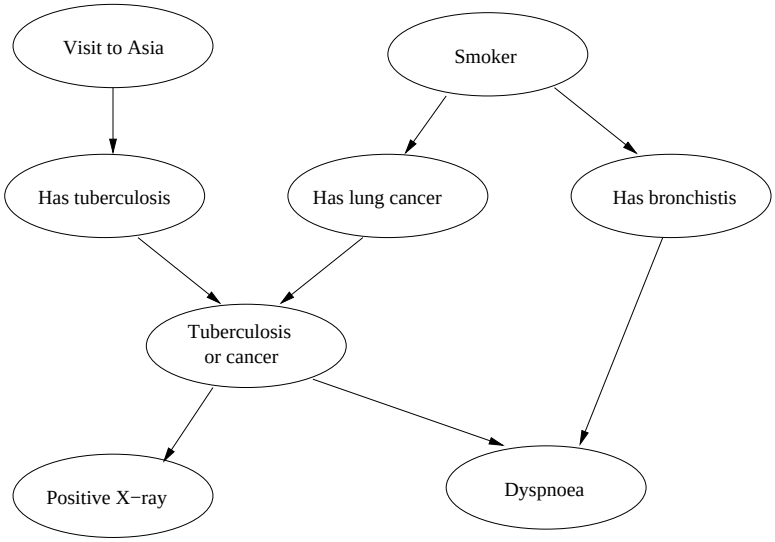

Figura 2.6: La red Asia.

ción normal univariada, se puede aplicar un método simple de suma de 12 variables uniformes independientes (31).

2.5 Redes utilizadas en los experimentos

En los experimentos de los capítulos 5 y 6 se han utilizado las redes *Asia* y *Alarm*. En el segundo de ellos se han utilizado además dos redes generadas artificialmente que están descritas en el propio capítulo. En los capítulos 4 y 7, dedicados a la triangulación, se han usado las redes *Sparse* y *Dense*.

La red Asia (ver Figura 2.6) fue creada por Lauritzen y Spiegelhalter (97) como ejemplo de red Bayesiana. Puede ser vista como una red destinada a realizar la diagnosis de pacientes. Tiene 8 variables, cada una de las cuales con 2 posibles estados, y 8 arcos. Los 3000 casos de la base de datos que se ha usado en el problema del aprendizaje estructural (capítulo 5) han sido generados con *Netica* de Norsys Software Corp. (46).

La red Alarm (ver Figura 2.7) es un estándar en la evaluación de algoritmos de aprendizaje. Su nombre está creado a partir de *a logical alarm reduction mechanism*, y fue propuesta por Beilinch y col. (16) con el propósito de crear un sistema de mensajes de alarma para diagnóstico médico que pudiera vigilar a los pacientes.

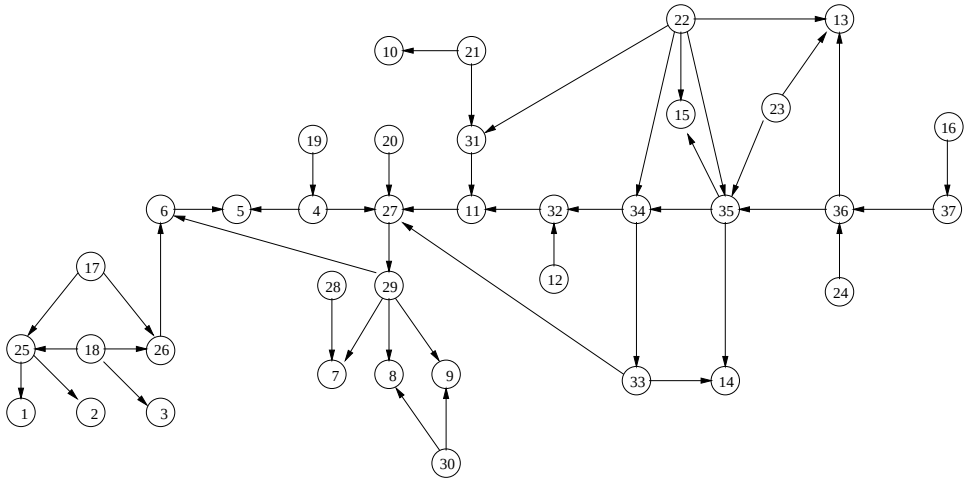

Figura 2.7: La red Alarm.

Tiene 37 variables (de 2, 3 y hasta 4 estados posibles) y 46 arcos. La base de datos de 3000 casos utilizada en el problema del aprendizaje estructural puede ser obtenida desde el almacén de bases de datos para aprendizaje automático de la UCI (111) (se han usado en los experimentos únicamente los 3000 primeros casos).

La red Sparse está generada por ordenador y tiene 50 variables y 100 arcos, mientras que Dense, también generada artificialmente, tiene 50 variables y 359 arcos. Ambas redes son de arcos no dirigidos, lo cual simplifica el proceso de triangulación, como se verá en el capítulo correspondiente, ya que no hace falta moralizar la red.

Capítulo 3

Introducción a los EDAs

'La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica.'

Aristóteles

En este capítulo se presentarán los algoritmos EDAs y se describirán aquellos que van a ser utilizados en esta memoria. Una vez establecidas las caraterísticas básicas de este tipo de algoritmos, podremos hablar de la representación del individuo EDA que usaremos en todos los problemas propuestos, así como de las redes Bayesianas estándar que servirán como contexto de la resolución de dichos problemas.

3.1 Algoritmos EDAs

EDAs son las siglas de *estimation of distribution algorithms* o algoritmos de estimación de distribuciones (88; 92; 110; 101). Se trata de un nuevo paradigma de la computación evolutiva. Como los algoritmos genéticos, los EDAs están basados en búsquedas estocásticas en poblaciones en las que se sustituye los operadores de cruce y mutación propios de los primeros por el aprendizaje de la distribución de probabilidad de los mejores individuos de cada generación y su posterior simulación. De esta manera, las relaciones entre las distintas variables son recogidas de manera explícita en el modelo.

3.1.1 Definiciones previas

Antes de mostrar el pseudocódigo de la aproximación EDA (92), estableceremos algunas definiciones previas:

- $\mathbf{z} = (z_1, \ldots, z_m)$ denotará la instanciación de la variable m-dimensional $\mathbf{Z} = (Z_1, \ldots, Z_m)$ llamada individuo de m genes.
- D_l será la población de M individuos en la generación l-ésima.
- D_l^s será la población de N individuos seleccionados a partir de la población D_l.
- $\rho_l(\mathbf{z}) = \rho_l(\mathbf{z}|D_{l-1}^s)$ será la función de densidad de probabilidad conjunta generalizada (o masa de probabilidad si cada Z_i es una variable aleatoria discreta) del individuo $\mathbf{z}$ en la generación l, dado D_{l-1}^s.

La forma general del algoritmo EDA propuesto en este trabajo puede ser vista en la Figura 3.1. Los grados de libertad de los que disfrutan estos algoritmos se limitan a:

- La obtención de la primera generación, que no tiene porqué ser forzosamente aleatoria.
- El método de selección de individuos.
- La estimación de la función de densidad $\rho_l(\mathbf{z})$, que a menudo lleva asociada la generación de un modelo gráfico probabilístico.
- La obtención de la nueva generación dada la estimación.
- El criterio de parada y, por lo tanto, la función de evaluación.

Para estimar $\rho_l(\mathbf{z})$ y simular posteriormente la nueva generación, los EDAs construyen y usan una red Bayesiana si las variables a partir de las cuales están construidos los individuos son discretas, y una red Gaussiana si son contínuas. La estimación de la función de densidad es la parte más dificultosa del EDA. Un buen resumen de las propuestas en los últimos años puede leerse en (92; 101).

```
Procedimiento Eda
  D_0 ← Generar M individuos aleatoriamente (la población inicial)
  l ← 1
  Repetir
    Para cada individuo i_i(i = 1, ..., M) hacer
      Evaluar el individuo i_i con una función de evaluación
      determinada sobre s_i
      Si s_i es la mejor evaluación hasta el momento entonces
        b_i ← i_i
      FinSi
    FinPara
    D^s_{l-1} ← Seleccionar N ≤ M individuos de D_{l-1} utilizando un
                método de selección prefijado
    ρ_l(z) ← Estimar la distribución de probabilidad de z
             dado D^s_{l-1}
    D_l ← Simular M individuos (la nueva población) a partir de
          ρ_l(z)
    l ← l + 1
  Hasta que un criterio de parada relacionado con la función de
       evaluación de los individuos sea satisfecho
  Devolver b_i
```

Figura 3.1: Pseudocódigo de un EDA.

3.1.2 La obtención de la primera generación

Se puede inicializar cada variable unidimensional con una distribución conocida, como la de Bernouilli, por ejemplo. En nuestro caso, sin embargo, optaremos por una población completamente aleatoria, excepto en el caso de la aproximación recursiva del capítulo 7, en la que se mantendrá una cache entre llamadas recursivas que trata de aprovechar los mejores órdenes relativos entre las variables obtenidos en invocaciones anteriores.

3.1.3 El método de selección

Evidentemente, tiene que tener forzosamente una relación con la función de evaluación, que es la que en realidad orienta al algoritmo en la búsqueda de la solución

más adecuada. En nuestro caso el grupo de individuos seleccionados está compuesto por aquellos que devuelven los mejores valores de la función de evaluación.

3.1.4 La obtención de la nueva generación

Tal y como se ha comentado en el capítulo precedente, en los experimentos con variables discretas (redes Bayesianas) se ha utilizado el *Probabilistic Logic Sampling* o PLS, método propuesto por Henrion (77). En aquellos que implican redes Gaussianas (variables contínuas) se ha utilizado una adaptación del método de condicionamiento, cuya descripción puede verse en el libro de Ripley (124), relacionada también con el PLS.

3.1.5 Estimación de la distribución de probabilidad en dominios discretos: independencia total

Obviamente, la obtención de todos los parámetros necesarios para especificar el modelo probabilístico es intratable. Por lo tanto, habremos de buscar otros métodos para conseguir una estimación, a veces de manera heurística, y otras simplificando el problema en relación a la fuerza de la independencia de las variables entre sí. En cualquier caso, siempre habremos de recurrir a los datos del conjunto de individuos seleccionado.

Como primera aproximación, se puede asumir que las variables aleatorias son independientes las unas de las otras, esto es, que hay independencia total entre ellas. Esto está lejos de recoger la idiosincrasia de muchos problemas combinatorios, evidentemente aquellos en los que determinadas variables están interrelacionadas unas con otras. En cualquier caso, es la aproximación más sencilla y proviene de los trabajos iniciales de Syswerda (147), Baluja (12), Eshelman y Schaffer (57) y Mühlenbein y Voigt (110) que partieron de la idea de conseguir un algoritmo basado en poblaciones que no dependiera de las operaciones de cruce y mutación específicas que lastran la generalidad de los algoritmos genéticos. En este sentido los EDA son más independientes (aunque no del todo) del problema a resolver, ya que carecen de dichos operadores, que han de estar seleccionados de entre los que mejor se comportan a la hora de resolver el problema. La *dependencia* de los EDA con respecto al problema se limita a la función de evaluación, dependencia que también tienen los algoritmos genéticos. En este caso, la distribución de probabilidad conjunta n-dimensional se factoriza como el producto de n distribuciones de probabilidad univariantes e independientes:

$$\rho_l(x) = \prod_{i=1,\ldots,n} \rho_l(x_i) \tag{3.1}$$

La generalización de esta idea propuesta por Münlenbein (107) dio lugar al algoritmo UMDA (*Univariate Marginal Distribution Algorithm*), que es uno de los EDA que utilizaremos en los experimentos. En este caso cada distribución marginal univariante se estima usando las frecuencias marginales:

$$\rho_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_{l-1}^s)}{Nn} \tag{3.2}$$

donde

$$\delta_j(X_i = x_i | D_{l-1}^s) = \begin{cases} 1 & \text{si en el } j-\text{ésimo caso de } D_{l-1}^s, X_i = x_i, \\ 0 & \text{en caso contrario.} \end{cases} \tag{3.3}$$

Véanse los trabajos de Santana y col. (131; 132) para posibles modificaciones de UMDA en la fase de simulación de la red. Hay otros algoritmos que también tienen como suposición a priori la independencia de las variables, como el *algoritmo genético compacto* de Harik y col. (74) o cGA para variables binarias, en el que tras una inicialización del vector de probabilidades usando una distribución de Bernouilli, dos individuos son generados aleatoriamente a partir de dicha distribución y, tras ser evaluados, comienza una competición entre ellos, ajustándose el elemento $i-$ésimo del vector de probabilidades (sumando o restando una constante) cuando el individuo perdedor toma un valor diferente del ganador en el $i-$ésimo gen, hasta que el vector converge. En el ya citado *aprendizaje incremental basado en poblaciones* o PBIL de Baluja (12; 13), se adapta el vector de probabilidades mediante una regla de cambio que se inspira en la regla de Hebbian usada en sistemas de redes neuronales.

3.1.6 Estimación de la distribución de probabilidad en dominios discretos: dependencias entre pares de variables

Si asumimos que las dependencias se reducen a pares de variables, el algoritmo se complica, ya que mientras que en la anterior sección la estructura se mantiene invariable, en esta cambia, ya que al modificar la estimación de la distribución estamos cambiando también la interdependencia de las variables unidimensionales, y habremos ya de hablar de modelos gráficos probabilísticos, y no de variables *aisladas*

gráficamente unas de otras. Es decir, no realizamos únicamente un aprendizaje paramétrico, sino también estructural. Limitando la dependencia a pares de variables, nos mantenemos en estadísticas de segundo orden. En este caso obtenemos como estructura un *grafo en cadena* en el que cada variable es hija de un solo padre y cada padre tiene un solo hijo (exceptuando los nodos de los extremos de la cadena). Esto nos fuerza a buscar una buena permutación en la que el grafo en cadena sepa recoger lo mejor posible la idiosincrasia del problema.

Una estrategia posible es hallar aquella permutación de las variables, de entre todas las permutaciones posibles, que hace que la distribución de probabilidad perteneciente a la clase de distribuciones $\mathrm{P}_\pi(\mathbf{x}) = \{\rho_\pi(\mathbf{x}) | \rho_\pi(\mathbf{x}) = \rho(x_{i_1}|x_{i_2}) \cdot \rho(x_{i_2}|x_{i_3}) \ldots \rho(x_{i_{n-1}}|x_{i_n}) \cdot \rho(x_{i_n})\}$ sea más cercana, con respecto a una determinada distancia, a la empírica extraída del conjunto de individuos seleccionados. $\pi = (i_1, i_2, \ldots, i_n)$ sería una permutación de los índices $1, 2, \ldots, n$. Estas distancias varían según los autores. De Bonet y colaboradores, en su algoritmo voraz MIMIC *Mutual Information Maximization for Input Clustering* (23) proponen la divergencia de Kullback-Leibler entre dos distribuciones $\rho_l(\mathbf{x})$ y $\rho_l^\pi(\mathbf{x})$. Esta divergencia es una función de:

$$H_l^\pi(\mathbf{x}) = h_l(X_{i_n}) + \sum_{j=1}^{n-1} h_l(X_{i_j}|X_{i_{j+1}}) \tag{3.4}$$

donde $h(\mathbf{X}) = -\sum_x \rho(\mathbf{X} = \mathbf{x}) \log \rho(\mathbf{X} = \mathbf{x})$ es la entropía de Shannon de la variable $\mathbf{X}$ y $h(\mathbf{X}|\mathbf{Y}) = \sum_x \rho(\mathbf{X} = \mathbf{x}) h(\mathbf{X}|\mathbf{Y} = \mathbf{y})$ la entropía condicional de $\mathbf{X}$ dado $\mathbf{Y}$, siendo $h(\mathbf{X}|\mathbf{Y} = \mathbf{y}) = -\sum_x \rho(\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y}) \log \rho(\mathbf{X} = \mathbf{x}|\mathbf{Y} = \mathbf{y})$ verificándose que $h(\mathbf{X}|\mathbf{Y}) = h(\mathbf{X}, \mathbf{Y}) - h(\mathbf{Y})$. Así pues, se trata de encontrar la permutación π^* que minimice $H_l^\pi(\mathbf{x})$. Buscar en las $n!$ posibles permutaciones puede ser prohibitivo. Para evitarlo, MIMIC busca esta mejor permutación seleccionando como la variable X_{i_n} aquella con menor entropía estimada (donde la estimación se realiza usando D_l^s) y, en cada paso, selecciona la variable cuya entropía condicional con respecto a la variable seleccionada en el paso anterior sea menor. Ésta es la aproximación que utilizaremos en los experimentos con variables discretas.

Otra opción es restringir la estructura del modelo, como en el *combining optimizers with mutual information trees* o COMIT de Baluja y Davies (14), haciendo que $\rho(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^n \rho(x_i|\pi_i^s)$, siendo π_i^s la instanciación del único padre de X_i, siendo todos los padres de todas las variables tal que no haya ciclos. Esto convierte la estructura en un árbol, representable mediante una red Bayesiana en la que las probabilidades asociadas se extraen del conjunto de individuos seleccionados y donde

la estructura se aprende utilizando el algoritmo de Chow y Liu (43). Esto facilita la búsqueda de la combinación de padres óptima para cada variable.

Otra aproximación es la de Pelikan y Mühlenbein (109) en su algoritmo BMDA o *Bivariate Marginal Distribution Algorithm*, en la que la estructura, además de estar compuesta por árboles, no es un grafo conexo (se trata de un grafo de dependencia). En esta propuesta se escoge una variable al azar, el padre, y se halla en el conjunto de individuos seleccionados aquella otra que tiene la mayor dependencia con respecto de ella (medida según la χ^2 de Pearson). A continuación se añade al grafo como segundo hijo una variable de entre aquellas no añadidas aún a la estructura, eligiendo la que más dependencia tiene con respecto a cualquiera de las que ya pertenecen al grafo. Estos dos pasos se repiten hasta que la dependencia no supera un determinado umbral, y este punto marca el final de la construcción de uno de los grafos del conjunto. El algoritmo se vuelve a aplicar al conjunto de variables que quedan sin añadir al modelo hasta que ya no queden variables por incorporar a éste. La estructura propuesta en este caso es un árbol binario que sólo desarrolla una de sus ramas.

3.1.7 Estimación de la distribución de probabilidad en dominios discretos: múltiples interdependencias

En este caso nos enfrentamos a factorizaciones de la distribución de probabilidad que implican estadísticas de orden mayor que dos.

En el *Extended Compact Genetic Algorithm* o EcGA de Harik (74), se parte, en cada generación, de una partición de las variables compuesta por n grupos (cada grupo contiene una única variable), entendiéndose que las variables de cada grupo son independientes de las del resto de los grupos. A continuación, un algoritmo de búsqueda intenta unir constantemente los dos clusters cuya unión disminuye una medida de la complejidad de la red: si esto no es así, la unión no se lleva a cabo, y este procedimiento se repite hasta que no hay grupos que puedan ser mezclados. De esta manera, la distribución de la probabilidad conjunta de las n variables queda como:

$$p_l(\mathbf{x}) = \prod_{g \in G_l} p_l(\mathbf{x}_g) \tag{3.5}$$

donde $\mathbf{x}_g$ denota una instanciación de las variables que pertenecen al grupo g. La partición del conjunto de variables G_l contiene grupos disjuntos de tamaño variable en cada generación. La medida que EcGA trata de minimizar en cada generación tiene dos componentes, la *complejidad compacta de la población*, que se define usando la

entropia de las distribuciones marginales y la *complejidad del modelo*, que se basa en el número de parámetros que se necesitan para especificar la distribución marginal de $\mathbf{X}_g$. La expresión es:

$$-N \sum_{g \in G_l} \sum_{\mathbf{x}_g} p(\mathbf{X}_g = \mathbf{x}_g) \log p(\mathbf{X}_g = \mathbf{x}_g) + \log N \sum_{g \in G_l} dim \mathbf{X}_g \tag{3.6}$$

expresión que Harik llama *complejidad combinada*.

Soto y col (143) utilizan una red Bayesiana con estructura de poliárbol en su algoritmo PADA, que puede considerarse una mezcla entre un método que detecta (in)dependencias condicionales y un algoritmo de búsqueda basado en métricas.

En el trabajo de Mühlenbein y col. (108) se presenta el FDA o algoritmo de distribución factorizada (*Factorized Distribution Algorithm*), que usa una distribución para ir generando poblaciones. Sea una partición $S = \{s_1, \ldots, s_k\}$ de $\{1, \ldots, n\}$. Si los siguientes tres conjuntos cumplen las correspondientes condiciones:

- $d_i = \bigcup_{j=1}^{i} s_j$ (historias)
- $b_i = s_i \backslash d_{i-1}$ (residuos)
- $c_i = s_i \bigcap d_{i-1}$ (separadores)
- $b_i \neq \varnothing \forall i = 1, \ldots, k$
- $d_k = 1, 2, \ldots, n$
- $\forall i \geq 2 \exists j < i$ t. q. $c_i \subseteq s_j$

entonces la función:

$$h(\mathbf{x}) = \sum_{s_i \in S} h_i(\mathbf{x}_{\mathbf{s_i}}) \tag{3.7}$$

se dice que es *aditivamente descomponible* (ADF). El FDA se aplica a las funciones aditivamente descomponibles, que trata de maximizar, para las que es capaz de obtener una factorización de la masa de probabilidad conjunta basada en los $\mathbf{x}_{b_i}$ y $\mathbf{x}_{c_i}$:

$$p_l(\mathbf{x}) = \prod_{i=1}^{k} p_l(\mathbf{x}_{b_i}|\mathbf{x}_{c_i}) \tag{3.8}$$

Estas masas de probabilidad pueden ser ya halladas en un tiempo polinomial y usadas para ir generando las distintas poblaciones. La desventaja de este algoritmo es la necesidad de trabajar siempre con ADFs.

En el algoritmo EBNA (*estimation of Bayesian networks algorithm* o algoritmo de estimación de redes Bayesianas) de Larrañaga y Etxeberria (88; 89) la distribución de probabilidad conjunta, que está codificada en una red Bayesiana, se aprende de la base de datos que contiene los individuos seleccionados en cada generación. Se comienza con un grafo acíclico dirigido sin arcos, y se les asigna a todos los puntos el espacio de búsqueda la misma probabilidad, creando la factorización de la distribución de probabilidad conjunta con n distribuciones uniformes univariadas. Para simular la red, se usa el PLS. A la hora de aprender la estructura, se prima la velocidad sobre la obtención de la red óptima, ya que se tiene que generar la estructura una vez por generación. Para ello se usa el algoritmo de búsqueda B (32), que en cada paso añade el arco que mayor incremento supone en la métrica. Dependiendo de la métrica utilizada tenemos diferentes versiones del mismo algoritmo (EBNA$_{K2}$ y EBNA$_{BIC}$), además de una en la que se realizan test de (in)dependencia condicional (EBNA$_{PC}$). Podemos encontrar numerosas aplicaciones en diferentes problemas y también una paralelización del algoritmo en (92). Un algoritmo parecido al EBNA$_{BIC}$ en el que se restringe el número de padres que puede tener un nodo puede verse en (109).

Pelikan y col. (119; 120) utilizan en el algoritmo BOA (*Bayesian Optimization Algorithm*) la métrica de la *equivalencia Bayesiana Dirichlet* o BDe, que devuelve el mismo valor si las redes reflejan el mismo conjunto de (in)dependencias condicionales, reduciendo además el número de posibles padres por nodo a k.

3.1.8 Estimación de la función de densidad en dominios continuos: independencia total

Aquí se asume que la función de densidad conjunta sigue una distribución normal n-dimensional que está factorizada por el producto de densidades normales unidimensionales independientes:

$$f_{\aleph}(\mathbf{x}; \mu, \sum) = \prod_{i=1}^{n} f_{\aleph}(x_i; \mu_i, \sigma_i^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}(\frac{x_i-\mu_i}{\sigma_i})^2} \tag{3.9}$$

La versión para los dominios continuos del algoritmo UMDA, que llamaremos UMDA$_c$, fue desarrollada por Larrañaga y col. en (90). A diferencia de la versión discreta, en este caso no podemos extraer directamente de la población la información paramétrica de las funciones de densidad unidimensionales. Se puede decir que hay un aprendizaje estructural, ya que las densidades unidimensionales del modelo son identificadas mediante tests de hipótesis. Tras ello, se halla el estimador máximo verosímil para realizar el aprendizaje paramétrico, tal y como vimos ya en el capítulo anterior en la sección dedicada a redes Gausianas. Este estimador tiene la siguiente expresión en el caso particular de que las densidades unidimensionales sigan distribuciones normales:

$$\widehat{\theta}_i^l = (\widehat{\mu}_i^l, \widehat{\sigma}_i^l) = \left(\frac{1}{N} \sum_{r=1}^{N} x_{ir}^l, \sqrt{\frac{1}{N} \sum_{r=1}^{N} (x_{ir}^l - \frac{1}{N} \sum_{r=1}^{N} x_{ir}^l)^2} \right) \tag{3.10}$$

Es decir, sólo tendremos que estimar la media y la desviación estándar de la normal.

El PBIL de Baluja y col. (12; 13) también tiene versión contínua, propuesta por Sebag y Ducolombier (135). En esta estrategia adaptativa, la evolución del vector de medias tiene la forma:

$$\mu_i^{l+1} = (1 - \alpha) \cdot \mu_i^l + \alpha \cdot (x_{i_{PrimeroMejor}}(l) + x_{i_{SegundoMejor}}(l) - x_{i_{peor}}(l)) \tag{3.11}$$

μ_i^l es el i-ésimo elemento de la media en la generación l y $x_{i_{PrimeroMejor}}(l)$ es el valor del i-ésimo elemento de $\mathbf{X}$ en el individuo de la población en curso (generación l) que mejor resultado da en la función de evaluación ($x_{i_{SegundoMejor}}(l)$ sería lo mismo pero referido al segundo mejor y $x_{i_{Peor}}(l)$ referido al peor individuo de la población actual). Para adaptar el vector de las varianzas los autores proponen varios heurísticos:

- Usar un valor constante para todas las probabilidades marginales y manterlo así durante todas las generaciones.
- Usar una estrategia evolutiva de tipo $(1, \lambda)$ (19).
- Calcular la varianza simple de los K mejores individuos de la generación en curso.
- Utilizar una regla de Hebbian similar a la usada para la evolución de las medias.

3.1.9 Estimación de la función de densidad en dominios continuos: dependencia entre pares de variables

Larrañaga y col. proponen en (88) y (90) una versión del algoritmo MIMIC para dominios continuos a la que llaman MIMIC$_c^G$. La idea consiste en usar en el modelo una única densidad marginal univariada y $n-1$ densidades condicionales bivariadas para definir la función de densidad conjunta, y tratarlas de ajustar lo máximo posible a las distintas poblaciones. Recordemos que en la versión discreta de MIMIC (23) se buscaba una permutación de las variables ideal usándose para evaluar la red resultante la entropía de Shannon. Larranaga y col. se basan para evaluar la red en el siguiente resultado de Whittaker (154) que dice que si $\mathbf{X} = \aleph(\mathbf{x}; \mu, \sum)$ es una función de densidad normal n-dimensional, entonces su entropía viene dada por la expresión:

$$h(\mathbf{X}) = \frac{1}{2} n(1 + \log 2\pi) + \frac{1}{2} \log |\sum| \tag{3.12}$$

Así, la expresión para la densidad univariada queda como:

$$h(X) = \frac{1}{2}(1 + \log 2\pi) + \log \sigma_X \tag{3.13}$$

y la de las bivariadas:

$$h(X|Y) = \frac{1}{2}\left[(1 + \log 2\pi) + \log \frac{\sigma_X^2 \sigma_Y^2 - \sigma_{XY}^2}{\sigma_Y^2}\right] \tag{3.14}$$

donde σ_X^2 es la varianza de la variable unidimensional X y σ_{XY} es la covarianza entre las variables X e Y. Para realizar el aprendizaje estructural parten en un primer momento de la variable Y con menor varianza de todas (que será la asociada a la densidad univariada) y a continuación van seleccionando, usando un algoritmo de búsqueda simple, la variable que será el próximo hijo. Para ello se elige aquella variable X aún no usada tal que el valor de $\frac{\sigma_X^2 \sigma_Y^2 - \sigma_{XY}^2}{\sigma_Y^2}$ sea el menor para así bajar la entropia, que es, como pasaba en la versión discreta, la expresión que se usa como distancia con respecto a la información que aparece en la población en curso.

3.1.10 Estimación de la función de densidad en dominios continuos: múltiples interdependencias

En este trabajo no utilizaremos este tipo de EDAs, pero aún así mencionaremos brevemente los más importantes. En este caso, dado que no se establece ninguna

restricción en la topología del grafo, tenemos una función de densidad multivariante que debe ser aprendida en cada generación.

En su algoritmo IDEA (24; 25), de *iterated density evolutionary algorithms*, Bosman y Thierens proponen reemplazar en cada generación solamente parte de la población y realizar la simulación de la red a partir de una distribución truncada mediante el peor individuo encontrado en la generación anterior. Como modelo a aprender en cada generación, usan, como en ejemplos anteriores, la normal multivariante, o bien la distribución del histograma o la distribución nuclear normal con la matriz diagonal de covarianzas. Para el aprendizaje usan un algoritmo de búsqueda basado en la distancia de Kullback-Leibler (26; 27) o bien un test de hipótesis (28).

En el algoritmo EMNA$_{global}$ (de *Estimation of Multivariate Normal Algorithm*) propuesto por Larranaga y col. en (90) se hallan los estimadores máximo verosímiles para el vector de medias $\mu_{\mathbf{l}} = (\mu_{1_l}, \ldots, \mu_{n_l})$, el vector de varianzas $\sigma_{\mathbf{l}} = (\sigma_{1_l}, \ldots, \sigma_{n_l})$ y la matriz de covarianza $\sum_l = \sigma^2_{ij_l}$ con $i, j = 1, \ldots, n$:

$$\widehat{\mu_{i_l}} = \overline{X_i^l} = \frac{1}{N}\sum_{r=1}^{N} x_{ir}^l \tag{3.15}$$

$$\widehat{\sigma_{i_l}^2} = \frac{1}{N}\sum_{r=1}^{N}(x_{ir}^l - \overline{X_i^l})^2 \tag{3.16}$$

$$\widehat{\sigma_{ij_l}^2} = \frac{1}{N}\sum_{r=1}^{N}(x_{ir}^l - \overline{X_i^l})(x_{jr}^l - \overline{X_j^l}) \tag{3.17}$$

para $i, j = 1, \ldots, n$ y $i \neq j$. Esto obliga a estimar, por generación $2n + \binom{n-1}{2}$ parámetros. Sin embargo, las estimaciones son matemáticamente sencillas.

Los mismos autores proponen el algoritmo EMNA$_a$, una versión adaptativa de EMNA$_{global}$. Mientras que el primer modelo $\aleph(\mathbf{x}; \mu_1, \Sigma_1)$ se estima a partir de los individuos seleccionados de la población inicial, luego el algoritmo se comporta como un algoritmo genético de estado estacionario. En cada paso se simula un individuo s^l a partir de la densidad normal multivariante. Si la evaluación del individuo es mejor que la del peor individuo p^l de la población actual, entonces el individuo simulado *sustituye* al peor en la población y la densidad se estima de nuevo mediante las siguientes expresiones (que se obtienen con métodos algebraicos):

$$\mu_{l+1} = \mu_l + \frac{1}{N}(s^l - p^l) \tag{3.18}$$

$$\sigma^2_{ij_{l+1}} = \sigma^2_{ij_l} - \frac{1}{N^2}(s^l_i - p^l_i) \cdot \sum_{r=1}^{N}(x^l_{jr} - \mu^l_j) - \frac{1}{N^2}(s^l_j - p^l_j) \cdot \sum_{r=1}^{N}(x^l_{ir} - \mu^l_i) +$$
$$\frac{1}{N^2}(s^l_i - p^l_i)(s^l_j - p^l_j) - \frac{1}{N}(p^l_i - \mu^{l+1}_i)(p^l_j - \mu^{l+1}_j) + \frac{1}{N}(s^l_i - \mu^{l+1}_i)(s^l_j - \mu^{l+1}_j) \quad (3.19)$$

EMNA$_i$ o EMNA incremental, de los mismos autores, también genera en cada paso solamente un individuo s^l a partir de la normal multivariante. Sin embargo, en este caso, el individuo siempre se añade a la población, pero sin quitar de ésta el individuo peor, con lo que la población va incrementándose constantemente (a diferencia del caso anterior, en el que su tamaño no variaba). La estimación de la nueva densidad se realiza ahora mediante las siguientes expresiones:

$$\mu_{l+1} = \frac{N_l}{N_l + 1}\mu_l + \frac{1}{N_l + 1}s^l \quad (3.20)$$

$$\sigma^2_{ij_{l+1}} = \frac{N_l}{N_l + 1}\sigma^2_{ij_l} + \frac{1}{N_l + 1}(s^l_j - \mu^l_j) \quad (3.21)$$

Al igual que EBNA, en el caso de los EDAs discretos, aprendía una red Bayesiana para codificar la distribucion de probabilidad conjunta, podemos usar una red Gausiana para hacer lo mismo. Esto es lo que proponen, de nuevo, Larrañaga y col. en (88; 90; 93) en la serie de algoritmos llamados EGNA. En (92) podemos ver diversas aplicaciones de estos EDAs debidas a Cotta y col. (ajuste de pesos en redes neuronales), Bengoetxea y col. (correspondencia de grafos) y Robles y col. (el problema del viajante). Tambien en (92) podemos ver una versión paralela debida a Lozano y col. La idea consiste en aprender la estructura de la red Gaussiana utilizando diversos métodos, como tests de exclusión de arcos (algoritmo EGNA$_{ee}$), búsquedas con métricas de tipo Bayesiano (EGNA$_{BGe}$) o con la de la máxima verosimilitud penalizada (EGNA$_{BIC}$). Después, es necesario estimar los parámetros de la red (el aprendizaje paramétrico). En el capítulo 2 se pueden encontrar más detalles sobre aprendizaje de redes Gausianas. Para simular la red y obtener la nueva generación, los autores usan una modificación del algoritmo PLS ya visto en el capítulo anterior.

3.1.11 Modelos mixtos

Tanto en el caso discreto como en el continuo, podemos agrupar de alguna manera las variables en grupos e interpretar que cada grupo es independiente de todos los

demás. La distribución de probabilidad conjunta se expresa entonces mediante un productorio de la distribución de probabilidad de cada grupo. Los grupos pueden ser fijos o cambiar junto con las poblaciones en cada generación, e incluso cada grupo puede tener asociado un peso. En la distribución de probabilidad de cada grupo puede haber ya múltiples interdependencias, que se modelizarán mediante una red Bayesiana, una función de densidad normal multivariante, etc. Veánse al respecto los trabajos de Pelikan y Goldberg (118) y Peña y col. en el capítulo IV de (92) para el caso discreto y los trabajos de Gallagher y col. (68; 67) y el mismo trabajo de Peña y col. para el caso continuo.

3.2 El criterio de parada

Normalmente, aquí se suele utilizar un criterio de convergencia, como por ejemplo que el porcentaje p de los genes de los nuevos individuos que devuelve el EDA cuyo valor es siempre el mismo sea superior a un determinado valor. Sin embargo, esto es difícil que ocurra en los problemas que afrontamos en este trabajo, relacionados en su mayor parte con búsquedas de órdenes óptimos, ya que a menudo el espacio de búsqueda es extremadamente grande. Aunque en cada capítulo utilizaremos un criterio de parada distinto, siempre estará relacionado con el número de *muestras*. Una muestra es una impresión que realiza el algoritmo del mejor individuo encontrado hasta entonces, cosa que hace cada cierto número de evaluaciones de individuos distintos. Este procedimiento nos permitirá seguir de manera sencilla la evolución del algoritmo. La decisión acerca del número de muestras y de cada cuántas evaluaciones se realiza una de ellas se lleva a cabo empíricamente. En algunos casos, los individuos propuestos por el EDA se repiten con una frecuencia tal que es necesario realizar la impresión de una muestra si se evalúan de manera consecutiva cierto número de individuos ya visitados.

3.3 Representación del individuo EDA

En el trabajo que nos ocupa, la mayor parte de los problemas están relacionados con un *orden* de las variables, esto es, buscamos un orden óptimo de éstas para conseguir un objetivo determinado. Debe, pues, haber una relación entre el individuo del EDA y un ordenamiento concreto de las variables. La evolución del EDA nos provee con diferentes individuos de m genes $(z_1, z_2, \ldots, z_m)$ que deben ser asociados unívocamente con un orden específico de las n variables.

3.3.1 Individuos continuos

En el caso de los individuos continuos, esta asociación es inmediata. Los genes del individuo tan solo han de ser ordenados. Después, a cada instanciación de gen se le asocia el índice que tiene en ese orden. El orden final de las variables es el individuo original en el que se sustituyen las instanciaciones de los genes por su índice. Por ejemplo, el individuo continuo $z_i = (0,5-1,6-0,2-0,1)$ queda, una vez ordenado, como $z_i' = (0,1-0,2-0,5-1,6)$. La asociación de índices queda como sigue: a la instanciación $0,1$ se le asocia el índice 1, a $0,2$ el índice 2, a $0,5$ el índice 3 y a $1,6$ el índice 4, de manera que sustituyendo instanciaciones por índices en z_i nos queda el orden de variables $(3-4-2-1)$.

Hay que tener en cuenta que la redundancia de esta representación es alta, puesto que diferentes individuos continuos pueden dar lugar al mismo órden de variables. Por ejemplo, el índividuo continuo $z_j = (3,2-5,0-1,2-0,5)$ se asocia al mismo orden que z_i.

3.3.2 Individuos discretos

Cuando los genes son discretos, la representación no es tan sencilla. Si tenemos 4 variables y 4! posibles permutaciones u órdenes, no podemos utilizar un individuo de 4 genes cuyas variables pueden tomar hasta 4 valores diferentes como mucho, ya que podemos llegar a obtener una instanciación del tipo $2-3-4-4$, que no es en absoluto un órden válido.

Esta dificultad puede ser soslayada penalizando los individuos que no verifican determinada restricción, en este caso los que den lugar a órdenes inválidos, o bien adaptando el proceso de simulación de nuevos individuos de manera que sólo proporcione individuos válidos (18). Sin embargo, ambos procedimientos tienen una influencia negativa en el comportamiento del EDA y traicionan de alguna manera la pretensión de generalidad de este tipo de soluciones.

Hay otra posibilidad que evita estos problemas y que además hace la asociación individuo-orden *biyectiva*, es decir, que además evita el problema de redundancia que aparecía en los individuos continuos. Podemos determinar un orden particular de los $n!$ posibles con la descomposición en factores de $n!$. Veamos un ejemplo. Si tenemos cuatro variables, los 4! posibles órdenes pueden ser generados de una manera sistemática, mostrada en el Cuadro 3.1. Vemos que primero desarrollamos todos los órdenes que tienen como primer nodo al nodo número 1. Dentro de éstos, que son 6 en este caso, procedemos de manera que formaremos primero aquellos que tienen como segundo nodo al nodo número 2, esto es, de los nodos que aún no hemos

Cuadro 3.1: Generación sistemática de los ordenes posibles para 4 nodos.

1 - 1234	7 - 2134	13 - 3214	19 - 4231
2 - 1243	8 - 2143	14 - 3241	20 - 4213
3 - 1324	9 - 2314	15 - 3124	21 - 4321
4 - 1342	10 - 2341	16 - 3142	22 - 4312
5 - 1432	11 - 2431	17 - 3412	23 - 4132
6 - 1423	12 - 2413	18 - 3421	24 - 4123

colocado, elegimos siempre el de índice más bajo. Cuando quedan únicamente dos nodos a y b sin colocar en el orden, es evidente que sólo tenemos dos posibilidades: ab y ba. Después comenzamos con los 6 órdenes siguientes, esto es, los que empiezan con el nodo de índice 2, y así sucesivamente hasta completar los 4! órdenes. La descomposición en factores de 4! es $4 \cdot 3 \cdot 2 \cdot 1$. Procederemos a asignar un gen del individuo a cada factor de la descomposición (menos al último). En este caso, pues, tendremos un individuo de 3 variables. Además, cada gen puede tomar tantos valores como el valor del factor, es decir, si el individuo de tres variables es (Z_1, Z_2, Z_3), tenemos que $r_1 = 4$, $r_2 = 3$ y $r_3 = 2$. El algoritmo que asocia a un determinado individuo un orden concreto funciona de la siguiente manera: básicamente mantiene una lista ordenada de manera ascendente de los nodos que aún no hemos colocado, y en cada paso determina, de entre los nodos que quedan en la lista, el que hay que colocar. En la Figura 3.2 podemos ver el algoritmo que realiza la asociación.

Por ejemplo, sea el individuo $(3, 2, 1)$. En un primer momento la lista de nodos sin colocar es $\{1, 2, 3, 4\}$. El primer gen del individuo puede tomar cuatro valores, así que es suficiente para decidir cual de los nodos hay que colocar en el orden: el de índice tres. Estamos pues en el grupo de órdenes $13 - 18$. La lista pasa a ser $\{1, 2, 4\}$, y ahora nos fijamos en el segundo gen, de valor 2. El gen tiene tres valores posibles, por lo que también es suficiente para decidir cuál de los tres nodos que nos quedan ha de ser colocado a continuación: el número 2. Estamos pues en el conjunto de órdenes $13 - 14$. El tercer y último gen nos dice que, a continuación, hemos de escoger el gen número 1 de la lista $\{1, 4\}$. Finalmente, sólo queda un nodo por colocar: el número 4, lo que da un orden final de $3 - 2 - 1 - 4$ (el orden número 13 de la lista sistemática).

En un problema con sólo 4 variables, tener genes con 4 valores posibles no resulta computacionalmente costoso. La cosa cambia si tenemos, por ejemplo, las 50 variables de la red Sparse. En este caso nos interesa más mantener un número de valores por

gen lo más pequeño posible, por ejemplo descomponiendo $n!$ en sus factores primos. Aún así, tendremos genes con un número de estados posibles grande (por ejemplo, 47 es primo). Esto complica además la determinación del orden a partir del individuo, ya que en cada paso nos hemos de preguntar por el número de genes que hacen falta para decidir cuál de las variables de la lista hay que colocar en el orden (ya no bastará con el gen en curso). Sin embargo, la ganancia que se obtiene mateniendo la mayor parte posible de genes con un número de estados bajo es grande, y en los experimentos de este trabajo se ha optado siempre por esta última opción.

Procedimiento **AsociaOrden** ($\mathbf{z} = \{z_1, z_2, \ldots, z_m\}$)
  $L \leftarrow \{Z_1, Z_2, \ldots, Z_m\}$
  $O \leftarrow \{\}$
  Para cada gen $Z_i (i = 1, \ldots, m)$ *hacer*
    $O \leftarrow O \bigcup \{l_{z_i}\}$, donde l_j denota
    el $j-$*ésimo* elemento de L
    $L \leftarrow L \setminus \{l_{z_i}\}$
  Fin Para
  Devolver O

Figura 3.2: Algoritmo de asignación individuo EDA - órden con los EDAs discretos.

Capítulo 4

Órdenes Óptimos en la Triangulación de Redes Bayesianas

'En la práctica sólo es problema lo que la inteligencia puede resolver.'

Hermann Keyserling

Uno de los usos más importantes de las redes Bayesianas consiste en realizar inferencias probabilísticas con ellas, esto es, conocer cuál es la instanciación más probable de un grupo de nodos dada una determinada instanciación de otro grupo. La inferencia resulta ser un problema NP-duro y, por tanto, computacionalmente costoso, ya que hay que realizarla mediante marginalización realizando una suma de probabilidades sobre las variables no instanciadas, trabajo que crece exponencialmente con el número de éstas y con el número de distintos valores que pueden tomar. Se han propuesto varios algoritmos para realizar inferencias aproximadas más eficientemente, como el uso de la programación dinámica o métodos de Monte Carlo. Uno de los más populares, y que además es exacto, es el de propagación de la evidencia de Lauritzen y Spiegelhalter (97), mejorado más tarde por Jensen y col. (81), y que básicamente trata de aprovechar las (in)dependencias de la red para que cuando el valor de un nodo cambia, sólo se vean implicados en un nuevo cálculo sus nodos vecinos y así no haya que tener en cuenta todos los nodos de la red en el mismo. Para la ejecución de este algoritmo es necesario establecer un orden previo de las variables para realizar la triangulación del grafo. La triangulación es un paso clave no sólo para realizar inferencias en una red Bayesiana, sino que también es importante en diseño VLSI o en problemas relacionados con bases de datos.

Este capítulo expondrá los resultados de la aplicación de los EDAs a la búsqueda

del mejor orden, esto es, aquel que minimiza el coste computacional de la inferencia mediante el algoritmo comentado. Dividiremos el capítulo en las siguientes secciones:

- En la Sección 4.1 se expone brevemente en que consiste el algoritmo de propagación de la evidencia de Lauritzen y Spigelhalter, poniéndose de manifiesto la necesidad de un orden óptimo.
- En la Sección 4.2 se exponen otros intentos de solucionar el problema mediante otros paradigmas de computación
- En la Sección 4.3 se proponen varios tipos de EDA para resolver el problema.
- En la Sección 4.4 se presentan los resultados obtenidos y se comparan con los de otros intentos de solucionar el problema.
- Finalmente, en la Sección 4.5 se exponen las principales conclusiones a las que los resultados obtenidos dan lugar.

4.1 Algoritmo de propagación de la evidencia de Lauritzen y Spigelhalter

Esta sección se ha confeccionado en base al artículo de Lauritzen y Spigelhalter (97). Para ilustrar el algoritmo, utilizaremos la red Bayesiana de la Figura 4.1. La descripción de los nodos, para un acercamiento más intuitivo, podría ser:

- D es la variable aleatoria *Existe Dios*. Puede tomar los valores d_1 (si) o d_2 (no). Como se puede ver por su distribución de probabilidad, hemos intentado no herir susceptibilidades.
- M es la variable aleatoria *Soy millonario*. m_1 es el valor asociado al si y m_2 el asociado al no.
- S es la variable aleatoria *Tengo salud*. s_1 significará que estoy sano y s_2 que estoy enfermo.
- F es la variable aleatoria *Soy feliz*. f_1 será el valor asociado a una respuesta afirmativa y f_2 el asociado a la negativa.

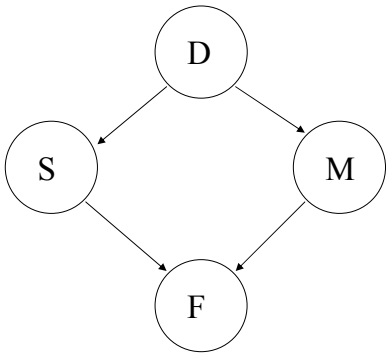

Figura 4.1: La red de ejemplo para saber si somos o no felices.

Cuadro 4.1: Distribución de probabilidad de D.

$P(D = d_1)$	$P(D = d_2)$
0,5	0,5

Cuadro 4.2: Distribución de probabilidad de M condicionada a D.

D	$P(M = m_1 \mid D = d)$	$P(M = m_2 \mid D = d)$
d_1	0,2	0,8
d_2	0,8	0,2

Cuadro 4.3: Distribución de probabilidad de S condicionada a D.

D	$P(S = s_1 \mid D = d)$	$P(S = s_2 \mid D = d)$
d_1	0,8	0,2
d_2	0,1	0,9

Cuadro 4.4: Distribución de probabilidad de F condicionada a M y S.

M	S	$P(F=f_1\|M=m,S=s)$	$P(F=f_2\|M=m,S=s)$
m_1	s_1	0,9	0,1
m_1	s_0	0,2	0,8
m_0	s_1	0,4	0,6
m_0	s_0	0,1	0,9

En los Cuadros 4.1, 4.2, 4.3 y 4.4, aparecen las distribuciones de probabilidad para cada una de las variables de la red. Vemos entonces que la distribución de probabilidad conjunta de una determinada instanciación de todas las variables es:

$$P(d,m,s,f) = P(d)\cdot P(m|d)\cdot P(s|d)\cdot P(f|m,s) \tag{4.1}$$

Introduzcamos ahora la evidencia de que soy feliz, esto es, F toma el valor f_1. ¿Cuál será el motivo más verosimil? ¿Ser millonario o tener salud? Podemos realizar esta inferencia utilizando la regla de Bayes:

$$P(M=m_1|F=f_1) = \frac{P(M=m_1,F=f_1)}{P(F=f_1)} = \sum_{d,s}\frac{P(D=d,M=m_1,S=s,F=f_1)}{P(F=f_1)} \tag{4.2}$$

$$P(S=s_1|F=f_1) = \frac{P(S=s_1,F=f_1)}{P(F=f_1)} = \sum_{d,m}\frac{P(D=d,M=m,S=s_1,F=f_1)}{P(F=f_1)} \tag{4.3}$$

donde $P(F=f_1) = \sum_{d,m,s} P(D=d,M=m,S=s,F=f_1) = 0,333$ es una constante que sirve para normalizar. Si hacemos los cálculos, descubrimos que $P(M=m_1|F=f_1) = 0,553$, mientras que $P(S=s_1|F=f_1) = 0,721$. Luego si soy feliz es más probable que sea debido a mi buena salud que a mi saneada cuenta corriente. El problema en las inferencias aparece cuando el número de variables no instanciadas (D, S y M en el ejemplo) empieza a crecer, pues el coste del cálculo crece exponencialmente con su número.

4.1.1 Potenciales de evidencia

Hallar el valor de $P(S = s_1|F = f_1)$ es afrontable en una red pequeña como la del ejemplo. Pero compliquemos algo más la red y consideremos la red Asia (ver Figura 4.2), en la que cada nodo puede tomar dos valores distintos. Esta red fue propuesta por Lauritzen y Spiegelhalter (97) y trata de representar las interrelaciones entre algunas enfermedades y sus síntomas. La descripción de los distintos nodos de la red es la siguiente:

- T significa *Tener tuberculosis.*
- L significa *Tener cancer de pulmón.*
- D significa *Tener disnea*, que como vemos puede estar provocada tanto por el cancer de pulmón como por la tuberculosis.
- E significa *Tener tuberculosis o cancer de pulmón.*
- X signigica *Resultado positivo en la prueba de rayos X en el pecho*, que como vemos puede ser debida tanto al cancer de pulmón como a la tuberculosis.
- B significa *Tener bronquitis*, que también puede producir disnea.
- A significa *Haber visitado recientemente Asia* (de ahí el nombre de la red), lo cual puede ser causa de la tuberculosis.
- S significa *ser fumador*, que puede conllevar la bronquitis o el cancer de pulmón.

La expresión de la probabilidad conjunta de una determinada instanciación de todas las variables es:

$$\begin{aligned} P(a,t,x,e,d,l,b,s) = \quad & P(a)\cdot P(t|a)\cdot P(x|e)\cdot P(e|t,l)\cdot \\ & \cdot P(d|e,b)\cdot P(l|s)\cdot P(b|s)\cdot P(s) \end{aligned} \tag{4.4}$$

En adelante, cambiaremos la notacion y haremos $P(A = a)$ equivalente a $P(a)$. Una pregunta sencilla como $P(x|a,d) = \frac{P(x,a,d)}{P(a,d)}$ implica el cálculo de 2^8 probabilidades conjuntas con su posterior suma. En redes con mayor número de nodos, evidencias que impliquen a un número de nodos pequeño y con más de dos valores posibles, una pregunta sencilla como ésta, de ser calculada por fuerza bruta, sería computacionalmente prohibitiva.

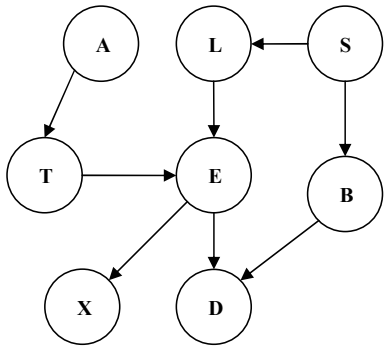

Figura 4.2: Estructura de la red Bayesiana Asia.

Sin embargo, en vez de calcular cada probabilidad conjunta por separado para sumarlas al final todas, podemos agrupar los factores comunes de todas ellas de manera que podemos hacer un poco más eficiente el cálculo:

$$P(a,x,d) = \sum_{t,e,l,b,s} P(a,t,x,e,d,l,b,s) = P(a)\sum_{t} P(t|a)$$
$$\left[\sum_{e} P(x|e)\left[\sum_{l} P(e|t,l)\left[\sum_{b} P(d|e,b)\left[\sum_{s} P(l|s)P(b|s)P(s)\right]\right]\right]\right] \tag{4.5}$$

No es necesario calcular la constante de normalización (la $P(a,d)$ en este caso), sino que podemos trabajar con valores directamente proporcionales hasta que sea necesario introducirla (53), lo cual simplifica aún más nuestros cálculos. Sin embargo, esto hace que cada factor no pueda ser interpretado como una probabilidad condicionada. Llamaremos a cada factor *potencial de evidencia*, y reescribiremos la ecuación 4.5 como:

$$\psi(A)\psi(T,A)\psi(X,E)\psi(E,T,L)\psi(D,E,B)\psi(L,S)\psi(B,S)\psi(S) \tag{4.6}$$

Las funciones de potencial de evidencia son, en un primer momento, las probabilidades condicionales, y se aplican a conjuntos de variables que se encuentran unidas en el grafo. Un procedimiento útil para poner de manifiesto las variables implicadas en cada potencial de evidencia es considerar el grafo no dirigido en el que unimos los padres de un hijo común. A este procedimiento se le llama *moralizar el grafo* (ver el capítulo 2). La moralización se aplica al grafo no dirigido: esto pone de manifiesto más fácilmente los nodos implicados en cada potencial. Para moralizar Asia, basta

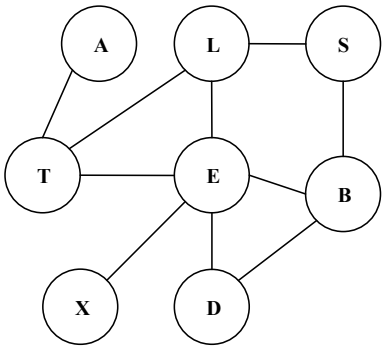

Figura 4.3: La red causal Asia no dirigida y moralizada.

con añadir los arcos no dirigidos $T - L$ y $E - B$ (ver Figura 4.3). Las funciones de potencial de evidencia se aplicarán así, a conjuntos de nodos completos (todos los nodos tienen un arco con el resto de los nodos del grupo).

Sin embargo, al sumar la expresión $\psi(L, S)\psi(B, S)\psi(S)$ para los distintos valores de S, obtenemos un nuevo factor (un nuevo potencial de evidencia intermedio) que depende de L y de B, que son nodos que no están unidos en el grafo moral: esto es un problema porque va en contra de la definición de la función de potencial. Podemos evitar este problema con el procedimiento al que se refiere este capítulo: la triangulación del grafo. Se dice que un grafo está triangulado cuando no hay ciclos de longitud mayor o igual que cuatro sin una cuerda. En la red Asia, podemos triangular el grafo con solo añadir el arco $L - B$ (ver Figura 4.4). En una red más compleja, triangular el grafo no será tan sencillo debido al número de posibilidades que tenemos (ver siguiente sección): será precisamente la triangulación el proceso más costoso de la *preparación* del grafo para una inferencia eficiente. Una buena triangulación puede hacer posible encontrar la solución de muchos problemas relacionados con grafos y NP-duros en tiempos polinomiales en lugar de exponenciales (9).

Una vez puesto de manifiesto los subgrafos completos que hay en el grafo, ¿cuáles escoger para establecer las funciones de potencial? Lo más adecuado va a ser aplicar las funciones sobre los cliques o conglomerados, esto es, sobre los subgrafos completos *maximales*, por lo que obtendremos una expresión tal que:

$$P(A, T, X, E, D, L, B, S) \propto$$
$$\psi(A, T)\psi(T, L, E)\psi(L, E, B)\psi(L, B, S)\psi(E, B, D)\psi(E, X) \tag{4.7}$$

donde

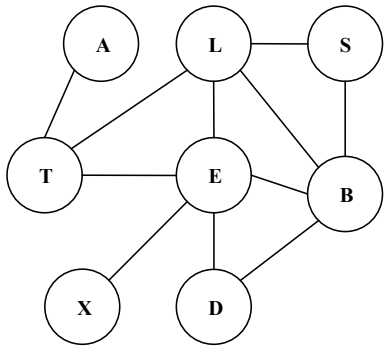

Figura 4.4: La red causal Asia no dirigida, moralizada y triangulada.

$$\begin{aligned}
\psi(A,T) &= P(A)P(T|A) \\
\psi(T,L,E) &= P(E|T,L) \\
\psi(L,B,S) &= P(L|S)P(B|S)P(S) \\
\psi(E,B,D) &= P(D|E,B) \\
\psi(E,X) &= P(X|E)
\end{aligned}$$

Esto es, reunimos en una las expresiones relacionadas con cada conglomerado de la ecuación 4.6: el potencial no es más que la marginal sobre el conglomerado correspondiente. Hay que tener presente que los potenciales se aplican sobre instanciaciones del conglomerado, es decir, que un conglomerado tendrá tantos potenciales de evidencia como instanciaciones posibles. Podremos notar esto de la siguiente manera:

- $\psi_{C_i}(\mathbf{c})$ es un potencial de evidencia del conglomerado C_i para una instanciación concreta de sus variables $\mathbf{c}$.
- $\psi(C_i)$ es el conjunto de los potenciales de evidencia para todas las instanciaciones posibles de C_i, que llamaremos por extensión el *potencial de* C_i.

Cuando hablamos entonces de $\psi_{C_i}(\mathbf{x}^i)$, donde $\mathbf{x}^i$ es una instanciación de un subconjunto de $\mathbf{X}$ que contiene a todas y cada una de las variables implicadas en C_i, es evidente que sólo utilizaremos en el cálculo del potencial el subconjunto de variables instanciadas que pertenecen al conglomerado C_i.

En general, para hallar los potenciales asociados a cada instanciación de un conglomerado C_i, primero se inicializan todos ellos a 1 y luego, para cada variable X_i de G, se busca un conglomerado C_j que contenga a toda la familia de X_i (si hay más de uno, se escoge al azar, por ejemplo el primero) y se multiplica $\psi(C_j)$ por $P(X_i|\Pi_i)$. En el caso de Asia, vemos que el conglomerado L, E, B da lugar a una función no definida que puede tomar un valor constante diferente de cero, por ejemplo 1: esto nos permitirá generalizar el algoritmo de propagación. Definir estas funciones sobre los conglomerados tiene una importante ventaja, ya que es posible expresar la distribución conjunta en función de las marginales sobre los conglomerados (97), lo que nos permitirá pasar del valor de las funciones de potencial a la probabilidad conjunta fácilmente. De hecho, en el caso de Asia, ésta última puede escribirse como:

$$\frac{P(A,T)P(T,L,E)P(L,E,B)P(L,B,S)P(E,B,D)P(E,X)}{P(T)P(L,E)P(L,B)P(E,B)P(E)} \tag{4.8}$$

Por tanto, almacenar todas las marginales de los conglomerados nos va a permitir realizar la inferencia de una manera eficiente. Para esto último, nos va a interesar definir la *marginalización de un grupo de variables* $\mathbf{I}$ *a otro* $\mathbf{J}$, donde $\mathbf{I} \subseteq \mathbf{J}$, como:

$$\psi_J(\mathbf{j})^{\downarrow I} = \sum_{J \setminus I} \psi_J(\mathbf{j}^*) \tag{4.9}$$

Donde la instanciación $\mathbf{j}^*$ resulta de copiar desde $\mathbf{j}$ las instanciaciones de las variables de $\mathbf{J}$ que también pertenecen a $\mathbf{I}$, y marginalizar sobre el resto. De igual manera podemos definir la *combinación de los potenciales de evidencia de dos grupos de variables* $\mathbf{I}$ *y* $\mathbf{J}$ para una instanciación $\mathbf{k}$ del grupo $\mathbf{K} = \mathbf{I} \bigcup \mathbf{J}$ como:

$$\psi_{I \bigcup J}(\mathbf{k}) = \psi_K(\mathbf{k})^{\downarrow I} \otimes \psi_K(\mathbf{k})^{\downarrow J} \tag{4.10}$$

Esto es, obtenemos un nuevo potencial que resulta de multiplicar punto a punto las marginalizaciones de K a I y J.

4.1.2 Construcción del árbol de conglomerados

Para realizar la propagación de la evidencia, después de hacer el grafo no dirigido, moralizarlo y triangularlo, se contruye el llamado árbol de conglomerados mediante el siguiente procedimiento:

- Numerar los nodos de la red, por ejemplo mediante el algoritmo de *búsqueda de la máxima cardinalidad* (149).

- Numerar los conglomerados usando el nodo de menor índice en el orden anterior. En caso de empate, se usa el siguiente nodo menor.

- Calcular los conjuntos R_i o *Residuales* y S_i o *Separadores*. Estos conjuntos se definen como siguen:

$$S_i = \begin{cases} C_i \bigcap (C_1 \bigcup \cdots \bigcup C_{i-1}) & \text{si } i \geq 2; \\ \varnothing & \text{si } i = 1. \end{cases}$$

$$R_i = \begin{cases} C_i \backslash S_i & \text{si } i \geq 2; \\ C_1 & \text{si } i = 1. \end{cases}$$

- Establecer la estructura del árbol, en el que los nodos serán los propios conglomerados, teniendo en cuenta que C_1 será la raíz del árbol y que un clique C_j podrá ser padre de un clique C_i si contiene al separador S_i (con $j < i$), pudiéndose elegir el padre aleatoriamente si hay más de uno. Como se vé, esto es un caso particular de grafo de conglomerados.

En un árbol así construido ya se puede realizar la propagación, ya que cumple las siguientes propiedades:

- Cualquier familia de un nodo del grafo se encuentra al menos en uno de los conglomerados.

- Dados dos conglomerados C_i y C_j, si $S = C_i \bigcap C_j \neq \varnothing$ entonces S está incluido en todos los conglomerados que se encuentran en el camino que une a C_i y C_j en el árbol de conglomerados, es decir, se trata de un *árbol de unión* (ver capítulo 2).

4.1.3 Propagación de la evidencia

El algoritmo de Lauritzen y Spiegelhalter que *propaga* la evidencia a través del árbol de m conglomerados $G' = (C, L')$, donde $C = \{C_1, \ldots, C_m\}$ es el conjunto de conglomerados y L' es el conjunto de aristas de dicho árbol, puede verse en la Figura 4.5. Se basa en dos fases, la ascendente en la que se recorre el árbol desde las hojas hasta la raíz (primer bucle) y tras lo cual tenemos que $\psi(G_i) = P(R_i|S_i)$; y la descendente en sentido contrario (segundo bucle) que hace los potenciales de cada conglomerado igual a su probabilidad conjunta, esto es, $\psi(G_i) = P(G_i)$. Finalmente,

Procedimiento **PropagaciónEvidencia** (G')
  $i \leftarrow m$
  Mientras $i > 1$ *Hacer*
    $\psi(S_i) \leftarrow \psi(C_i)^{\downarrow S_i}$
    $\psi(C_i) \leftarrow \frac{\psi(C_i)}{\psi(S_i)}$
    $\psi(\Pi_{C_i}) \leftarrow \psi(\Pi_{C_i}) \otimes \psi(S_i)$
    $i \leftarrow i - 1$
  FinMientras
  $i \leftarrow 2$
  Mientras $i \leq m$ *Hacer*
    $\psi(S_i) \leftarrow \psi(\Pi_{C_i})^{\downarrow S_i}$
    $\psi(C_i) \leftarrow \psi(C_i) \otimes \psi(S_i)$
    $i \leftarrow i + 1$
  FinMientras
  $i \leftarrow 1$
  Mientras $i \leq n$ *Hacer*
    Sea C_j el menor conglomerado que contenga a X_i
    $P(X_i) = \psi(G_j)^{\downarrow X_i}$
    $i \leftarrow i + 1$
  FinMientras
  Devolver $P(X_i) \forall i = 1, \cdots, n$

Figura 4.5: Algoritmo de propagación de la evidencia de Lauritzen y Spiegelhalter.

en el tercer bucle se calcula la distribución de probabilidad generalizada de cada variable.

Se puede utilizar el mismo algoritmo para el caso de que el valor de algunas variables ya se conozca, situación en la cual hay que realizar lo que Lauritzen y Spiegelhalter llaman *absorción de la evidencia* (es decir, para cada variable hallamos ahora $P(X_i|\mathbf{e})$ donde $\mathbf{e}$ es la evidencia, una instanciación concreta de un grupo de variables $\mathbf{E}$. Básicamente se trata de modificar el árbol eliminando las variables de $\mathbf{E}$ para que ahora factorize $P(\mathbf{X} \backslash \mathbf{X_0}, x_0)$.

Posteriormente, Jensen y col (81; 82) han mejorado este algoritmo de propagación de la evidencia (con un refinamiento ulterior en el algoritmo Hugin (7)) consiguiendo que no sea necesaria la modificación del árbol para realizar la absorción de la evidencia a costa de incrementar su tamaño al añadir como nodos a los separadores S_i en lo

que se llama el árbol de juntura o de intersecciones. Otro método de propagación es el de Shenoy y Shafer (140), que puede ser utilizado con distintos formalismos y en el que se parte de un árbol de juntura binario. Las ventajas de este método consisten, entre otras, en que no se realizan divisiones y las marginales para las variables de interés se calculan en grupos de variables que sólo contienen a las mismas, por lo que no es necesario marginalizar. Flores propone en (60) una mejora para hacer este método más eficiente con su *compilación incremental* de redes Bayesianas. Ninguno de estos métodos, sin embargo, aprovecha independencias inducidas por la propia evidencia o toma en consideración la dirección de las aristas en el grafo dirigido original y sus d-separaciones. Madsen y Jensen proponen en (102; 103) un método llamado *propagación perezosa* que sí lo hace, haciendo que la estructura en árbol sea menos compleja y por tanto que la propagación sea más eficiente.

En cualquier caso, el paso de la triangularización sigue siendo necesario para coordinar la propagación de la evidencia, coordinación que desaparece si hay un ciclo mayor o igual a cuatro en el grafo moral. La eficiencia de la triangulación mediatiza la de la propagación porque determina el tamaño de los conglomerados y la topología del árbol de juntura.

4.1.4 La triangulación

En el algoritmo con el que se realizarán los experimentos sobre triángulación se utilizará el conocido método de la eliminación de vértices o nodos. Informalmente hablando, cuando se elimina un vértice, se añaden tantos arcos al grafo original como sea necesario para que sus nodos adyacentes formen un subgrafo completo, y después se borra el vértice y todos los arcos relacionados con él. El grafo que incorpora todos los arcos nuevos al grafo original está triangulado. Es necesario, pues, establecer una ordenación α de los vértices antes de proceder a su eliminación. Es esta secuencia de eliminación la que determina la eficiencia de la estructura resultante, que viene dada por el tamaño de sus conglomerados: cuanto mayor sea el conglomerado, y mayor el número de estados distintos de sus integrantes, más grande será la tabla de probabilidad marginal (o la tabla de creencia en el caso de Hugin) asociada al conglomerado y más complejos serán los cálculos. Hay varios métodos para evaluar la calidad de una triangulación (129). Uno de los más comunes es el del peso mínimo, que es el que utilizaremos más adelante como función de evaluación del individuo en el EDA. Se asigna así un peso a cada conglomerado:

$$w(C_i) = \sum_{X_j \in C_i} log_2(r_j) \tag{4.11}$$

donde r_j es el número de estados diferentes del nodo X_j perteneciente al conglomerado C_i. Así, podemos definir también un peso para el grafo triangulado:

$$w(G^t) = log_2 \sum_{C_i \in C} \prod_{X_j \in C_i} r_j \tag{4.12}$$

Nos interesa pues hallar la secuencia de borrado de vértices que minimice $w(G^t)$, esto es, hallar el órden óptimo de los nodos que cumpla esa propiedad.

El método de borrado de vértices no es el único que se ha propuesto para realizar la triangulación de un grafo. Se trata de un problema NP-duro (8; 151) cuya solución por fuerza bruta implica tantas pruebas como posibles ordenaciones de n elementos (donde n es el número de nodos del grafo). En redes como *Alarm*, de 50 nodos, el intento por fuerza bruta implica 50! permutaciones, imposible de afrontar computacionalmente. En la siguiente sección veremos por un lado los intentos de obtener el mejor orden para el método del borrado de nodos utilizando distintos paradigmas de computación, así como otros métodos no relacionados con el borrado de nodos.

4.2 Otros intentos de minimizar el peso del grafo triangulado

Dividiremos esta sección entre aquellos métodos que tienen que ver con el borrado de los vértices, y por lo tanto con los que el nuestro compite directamente, y aquellos que utilizan otras estrategias.

4.2.1 Estrategias no basadas en el borrado de vértices

En los trabajos de Robertson y Seymour (125) y Becker y Geiger (15), no se busca un grafo triangulado que minimice el peso, sino aquel cuyo número de conglomerado sea como mucho $k + 1$, dado un entero k, ya que se consideran nodos binarios que, por tanto, tienen todos ellos el mismo peso. El número de conglomerado de un grafo es el tamaño del conglomerado más grande. Buscar esto es equivalente a encontrar la descomposición de un grafo cuyo tamaño sea como mucho k (ver capítulo 2). En los dos trabajos citados se utiliza la noción de α-vértice-separador. Dado un grafo

$G(X, L)$, un número real $\alpha \in (0, 1)$ y un conjunto $Y \subseteq X$, se dice que $Z \subseteq X$ es un α-vértice-separador de Y si todo componente conectado de $G[X \backslash Z]$ tiene al menos $\alpha \cdot |Y|$ vértices de Y. Z es un α-vértice-separador de dos vías si además hay dos conjuntos K_1 e K_2 tales que X los separa a ambos, $K_1 \cup K_2 \cup Z = X$ y $|K_i| \leq \alpha \cdot |Y| (i = 1, 2)$. Finalmente, Z será un (a, b)-vértice-separador del grafo no dirigido G si para todo $a, b \in V \backslash Z$, todo trayecto que conecte a y b en G pasa por lo menos por un vértice de Z. Un (a, b)-vértice-separador Z de G es de cardinalidad mínima si no contiene un subconjunto que también sea un (a, b)-vértice-separador. Mientras en (125) se intenta encontrar α-separadores de dos vías con distintos valores de α de manera recursiva, en (15) se considera en cambio la búsqueda del separador de n vías, en el que Z ahora separa $K_1, K_2, ...K_n$ conjuntos, en concreto el de 3. Leighton y Rao (98) y Amir (6) proponen pequeñas modificaciones de los algoritmos anteriores orientadas a hacerlos más eficientes. Estos algoritmos son interesantes porque son capaces de asegurar una solución aproximada a la óptima por un valor constante (o descubrir que esta solución no existe), aunque en pruebas prácticas consiguen descomposiciones de árboles peores y bastante más lentas que, por ejemplo, el *mindgree* (ver la siguiente sección).

Won y col. (155) han encontrado similitudes entre el modelo de redes Bayesianas y el modelo relacional de bases de datos. Hay una correspondencia biunívoca entre un hypergrafo y un esquema de base de datos relacional. Los nodos del hipergrafo son los atributos del esquema, mientras que la unión de los grupos de nodos que forman los hiperarcos es el esquema propiamente dicho, esto es, cada hiperarco se puede entender como un esquema de relación. Así, podemos decir que un esquema de base de datos relacional es acíclico si lo es el hipergrafo correspondiente (para más información acerca de los hipergrafos, ver capítulo 2). Se da el caso de que existe una correspondencia biunívoca entre el conjunto de grafos triangulados y el conjunto de hipergrafos acíclicos. Por cada grafo triangulado G podemos obtener un hipergrafo equivalente acíclico H donde cada hiperarco es un conglomerado maximal de G, y a la inversa, por cada hipergrafo acíclico H podemos obtener un grafo triangulado G que tiene los mismos nodos que H y un arco entre cada par de nodos que están en el mismo hiperarco de H. Los hipergrafos acíclicos son interesantes desde el punto de vista del modelo relacional de bases de datos porque los esquemas acíclicos de bases de datos asociados poseen unas propiedades interesantes para este modelo, por ejemplo que un hipergrafo acíclico es equivalente a un conjunto de relaciones llamadas de dependencia multivaluada (DMV) libre de conflicto. Existen algoritmos eficientes para construir un hipergrafo a partir de un conjunto de este tipo de relaciones. En realidad, hay una relación directa entre la definición de DMV y la de U-separación

$I(X, Y \backslash Z)$ de dos subconjuntos de nodos X e Y de un grafo G por otro subconjunto de nodos Z del mismo llamado *llave* (ver capítulo 2), por lo que podemos hablar de U-separaciones o independencias libres de conflicto y construir un hipergrafo acíclico a partir de dicha lista de independencias. Esta lista habrá de ser maximal para que no haya arcos superfluos en el grafo triangulado correspondiente.

Bodlaender y col. (21) afrontan el problema intentando, no ya encontrar un algoritmo eficiente que consiga una buena triangulación, sino tratando de preprocesar el grafo para obtener uno de tamaño más pequeño que sea, por tanto, más manejable pero que no nos haga perder la posibilidad de extrapolar el resultado al óptimo para el grafo inicial. La reducción del grafo se realiza en base a unas *reglas de reducción* reversibles (lo que nos permite la triangulación del grafo inicial a partir de la triangulación del grafo reducido). Las reglas son del tipo *si V es un vértice simplicial con un grado $d \leq 0$, entonces borrar V y actualizar el grado máximo encontrado (ley del vértice simplicial)*, donde un vértice de G es simplicial si sus vecinos forman un conglomerado en G, y su grado es el tamaño de este conglomerado. Otra regla, que refleja el hecho de la propia triangulación, es *si V es un vértice casi simplicial de G y su grado $d \leq max$, donde max es el máximo grado encontrado hasta el momento, entonces añadir un arco entre cada par de nodos no adyacentes en los vecinos de V y borrar V (ley del vértice casi simplicial)*. El algoritmo aplica distintos grupos de reglas de reducción al grafo moralizado hasta conseguir el grafo vacío o hasta que ninguna regla se pueda aplicar. Si este preproceso no consigue el grafo vacío, entonces se usa otro algoritmo que encuentre una triangulación cercana al óptimo del grafo reducido (si el grafo es pequeño, incluso se pueden usar métodos exactos, aunque la triangulación del grafo completo no lo será dado el caracter no exacto de la reducción). Sin embargo, este procedimiento tiene en cuenta solamente el mínimo tamaño del máximo conglomerado, y no los diferentes valores que puede tomar cada vértice.

Darwiche y Hopkins (49) trabajan sobre el balanceo de un d-árbol, ya que a partir de un d-árbol se puede obtener un arbol de juntura o un orden de borrado de nodos. Un d-árbol para un grafo G es un árbol binario completo cuyas hojas corresponden a las familias del grafo, y cuyos nodos que no son hoja sirven para realizar un particionado de las familias de G en dos ramas (y sucesivamente hasta llegar a los nodos hoja). Esta estructura también se puede utilizar directamente para realizar razonamientos lógicos, y las características que determinan su eficiencia son la altura y la anchura del árbol. Los autores proponen un algoritmo para crear d-árboles balanceados a partir del particionamiento de un hipergrafo en partes del mismo tamaño minimizando el número de hiperarcos que van de una parte a otra.

La idea es, pues, generar el hipergrafo equivalente al grafo dado, particionarlo, y luego convertirlo en el *d*-árbol (y transformar luego éste en un orden de borrado, o en un árbol de juntura). Esta estrategia produce resultados competitivos con el heurístico del mínimo número de arcos.

4.2.2 Estrategias basadas en el borrado de vértices

En cuanto a las estrategias relacionadas con el borrado de vértices, una posibilidad es realizar una búsqueda exhaustiva sobre el espacio de órdenes sobre las variables, procedimiento que nos daría la triángulación óptima. Sin embargo esto es imposible con conjuntos de nodos grandes. Una posibilidad que funciona con grafos de tamaño moderado es dejar de investigar un orden con un prefijo $(V_1, V_2, \ldots, V_k)$ concreto si en uno ya investigado con ese prefijo alcanzamos una suma de pesos de los conglomerados generados hasta ese momento superior al mínimo almacenado (algoritmo de *branch-and-bound*). Para grafos grandes no hay más remedio que recurrir a los heurísticos. Tenemos tres evidentes según Kjaerulff (84):

- H1 - El que borra en cada paso aquella variable no previamente borrada que produce el conglomerado máximo más pequeño (también llamado *Mindgree*).
- H2 - El que borra la variable que minimiza el número de arcos añadido.
- H3 - El que escoje la variable que minimiza el peso completo del grafo triangulado.

Ninguno de los tres algoritmos asegura los tres mínimos al mismo tiempo. El algoritmo H1 es el más rápido de los tres, implementable, dado $G(X, L)$, en $O(|X| + |L| + |T|)$, donde T es el número de arcos añadido tras la triangulación, y aunque en general produce buenos resultados, no tiene porqué producir el óptimo, ya que el borrar determinada variable con el conglomerado más pequeño puede producir en futuros pasos conglomerados más grandes. Además, según Rose (129), no produce necesariamente un buen orden de borrado si el grafo ya está triangulado.

Cano y col. (37) proponen una serie de algoritmos, modificaciones del H1, en los que se tiene en cuenta también el número de posibles valores de la variable o vértice a borrar, el incremento en tamaño del conglomerado más grande en el que está incluida cada variable, o el de la suma de todos los conglomerados en los que está incluida la misma. En (70), Gámez y Cano proponen un algoritmo de optimización basado en colonias de hormigas (22; 55). En estos algoritmos multiagente, cada agente (hormiga) deposita *feromona* en un suelo virtual durante su búsqueda del mejor trayecto

hacia el alimento (que en este caso es la mejor secuencia de borrado). Toda la colonia está influenciada por este comportamiento, ya que el resto de las hormigas toman decisiones basadas en la cantidad de feromona que se encuentra depositada sobre las diferentes ramas o caminos. En este trabajo, cada hormiga usa el resultado de uno de los algoritmos citados en (84) y (37) para construir su solución, obteniéndose buenos resultados con una velocidad mayor que la de otros métodos de optimización.

Todos estos algoritmos hablan directamente del borrado de vértices a la hora de buscar la mejor triangulación, pero no se basan en un ordenamiento previo de las variables. En este capítulo queremos afrontar, como sucederá en el capítulo 5, una búsqueda en el espacio de órdenes, por lo que nos hará falta un algoritmo que, dado un orden concreto, intente aproximarse a la mejor triangulación basándose de alguna manera en él.

Rose y col. (130) con el algoritmo de búsqueda lexicográfica y Tarjan y Yannakakis (149) con su algoritmo de búsqueda de la máxima cardinalidad, intentan separar los vértices de manera que no haya caminos entre ellos y se basan en una ordenación previa de los mismos. El primero elige en cada paso el nodo que maximiza la distancia lexicográfica; almacenando la lista de los nodos adyacentes numerados para cada nodo como si fuera una variable de cadena, se halla esta máxima distancia teniendo en cuenta que la de V es mayor que la W si su lista es lexicográficamente mayor. Los segundos eligen en cada paso el nodo que maximiza la cardinalidad, definiendo ésta como el mayor índice de los vecinos de un nodo que han sido ya eliminados. Sin embargo, éstas técnicas tienden a añadir arcos innecesarios. Fujisawa y Orino (64) proponen una técnica de eliminación extendida que evita este hecho, poniendo en evidencia que, dado un grafo $G(X, L)$, si los vecinos de un vértice V forman junto con V todo X, o los vecinos de V separan G en varios componentes, basta con hallar una triangulación de $G\backslash V$ en el primer caso o hallar las triangulaciones de los componentes en los que se separa G sin necesidad de añadir arcos a la hora de borrar V.

Kjaerulff (84) propone la búsqueda de un orden pero sobre los arcos añadidos en una triangulación, ya que utiliza un algoritmo recursivo de borrado de arcos innecesarios que se basa en un orden previo de ellos: el grafo ya está triangulado siguiendo algún heurístico previo, por ejemplo el de búsqueda de la máxima cardinalidad, y de lo que se trata es de minimizar el número de arcos que se añaden.

El mismo autor propone en (85) una búsqueda en el espacio de órdenes pero ya sobre los vértices del grafo, tratando el problema mediante *enfriamiento estadístico*. En este caso el espacio de soluciones S para el algoritmo será, dado el grafo $G(X, L)$, el conjunto de posibles órdenes de X, y la perturbación de un orden α_i que provoca

la transición a otro distinto α_j consiste en intercambiar dos vértices en α_i. El enfriamiento estatístico es un algoritmo estocástico que trata de minimizar una función de coste que en nuestro caso será el peso del grafo triangulado resultante de aplicar el borrado implícito en el orden. Este algoritmo evita los mínimos locales haciendo que el coste sea aceptado como mínimo, aún no siendo tal, con una probabilidad que disminuye con el tiempo y que depende de un factor t llamado temperatura. t puede ir decrementándose con el tiempo, de hecho así está descrito por sus creadores (ver Kirkpatrick y col. (83)) pero Kjaerulff consigue los mejores resultados para una temperatura baja y constante y eligiendo los vértices para las transiciones de manera que el radio de S (el máximo número de transiciones entre dos configuraciones distintas de S y que depende de las perturbaciones permitidas) se incremente poco a poco. Aunque este algoritmo nos da la triangulación mínima con una probabilidad tan cercana a 1 como queramos, en la práctica ese acercamiento es computacionalmente muy costoso, y de los tres parámetros sobre los que podemos actuar para decrementar el número de interaciones (la temperatura inicial, el ratio de enfriamiento y la temperatura final), hemos de sacrificar la temperatura final, esto es, aumentar el error permitido.

Wen (150) propone utilizar enfriamiento estadístico con dos métodos de perturbación distintos a los de Kjaerulff. Primero utiliza un heurístico para triangular el grafo, y toma como estados (órdenes) entre los que puede darse una transición, aquellos que el algoritmo ha probado consecutivamente, por un lado, o aquellos que resultan de intercambiar variables adyacentes (más restrictivo que en el caso de Kjaerulff) en el orden de partida. Como pasaba en (85), hemos de sacrificar la velocidad del algoritmo en aras de una mejora del resultado, y el problema está en hasta qué punto anula este sacrificio las ventajas de simulated annealing.

Meila y Jordan (104) consideran una función de coste que no tiene en cuenta sólo los conglomerados. En este caso tenemos que, dado un grafo $G(X, L)$, $w(\alpha) = \sum_{x \in X} \prod_{l_j \in C_X} n_j$, donde $w(\alpha)$ es el coste del orden α y C_X tiene la expresión:

$$C_X = \{X \cup V \in adj(X) | \alpha(X) > \alpha(V)\} \tag{4.13}$$

Con esto intentan simplificar el cálculo del coste. Por otra parte, integran $w(\alpha)$ en una funcion contínua en la que se penalizan los órdenes no válidos, y resuelven la tarea de optimización de dicha función obteniendo resultados similares al *Mindegree*.

Larrañaga y col. (94) buscan en el espacio de órdenes con algoritmos genéticos. La manera de resolver el problema tiene mucha relación con la que utilizaremos en este capítulo, no en vano ambos tipos de algoritmos pertenecen a los llamados *evolutivos*, en los que una población inicial de individuos evoluciona globalmente hacia otra

incorporando (generando) nuevos individuos y *matando* a otros que se consideran no adecuados para resolver el problema en cuestión. En la generación cooperan uno o más individuos de la población (ver una descripción más formal en el Capítulo 3). El individuo, en este caso, es directamente un órden. A una población inicialmente aleatoria se le aplica, para generar la generación siguiente, una serie de operadores de mutación y cruce. Sólo los individuos con un buen coste, que en este caso es $w(G^t)$, *sobreviven* de una generación a la siguiente. El proceso evolutivo se detiene cuando se alcanza la convergencia en algún sentido al estar todos los individuos de la población muy próximos entre si (por ejemplo usando un porcentaje de genes, nodos del orden, iguales). Aunque los algoritmos genéticos dan un buen resultado, exigen un estudio previo bastante costoso dada la cantidad de parámetros de los que dependen, y normalmente son aquellos que combinan operadores de cruce y mutación más costosos los que mejor resultado proporcionan.

4.3 Solución del problema con EDAs

Antes de seguir con las características de los EDA concretos que aplicaremos a este problema, trataremos de formalizarlo brevemente.

4.3.1 Formalización del problema

Definición 4.3.1 *Denotaremos como G_α al grafo $G = (\mathbf{X}, \mathbf{L})$ cuyo conjunto de nodos $X = \{X_1, \ldots, X_n\}$ ha sido ordenado mediante la biyección α.*

Definición 4.3.2 *Llamaremos peso de un nodo X_i a $w(X_i) = \log_2(r_i)$, donde r_i es el número de estados posible del nodo.*

Definición 4.3.3 *Llamaremos conjunto de nodos monótonamente adyacentes a un nodo X_i en el grafo ordenado G_α a $mady(X_i, G^\alpha) = adj(X_i, G_\alpha) \cap \{X_j \in X | \alpha(X_i) < \alpha(X_j)\}$.*

Definición 4.3.4 *Diremos que un nodo X_i ha sido eliminado del grafo $G = \{X, L\}$ si se añaden tantas aristas a G como sean necesarias para que $ady(X_i)$ sea un conglomerado y se quitan después X_i de X y las aristas incidentes en X_i de L.*

Definición 4.3.5 *Definiremos $G_{\alpha,\sigma} = (\mathbf{X}^\sigma, \mathbf{L}^\sigma)$ al grafo que se obtiene tras la eliminación de los vértices $\alpha(1), \alpha(2), \ldots \alpha(\sigma)$ en este orden.*

Evidentemente, $G_{\alpha,0} = G_\alpha$.

Definición 4.3.6 *Definiremos* $X^\sigma = \{X_i \in \mathbf{X} | \alpha(\mathbf{X_i}) > \sigma\}$.

Aquí tenemos que $X^0 = X$ y $X^n = \emptyset$.

Definición 4.3.7 *L^σ se define como:*

$$L^\sigma = \{\{X_i, X_j\} \in L^{\sigma-1} | X_i, X_j \in X^\sigma\} \cup \{\{X_i, X_j\} | X_i, X_j \in mady(\alpha(\sigma), G^{\alpha,\sigma-1})\}.$$

Donde $L^0 = \mathbf{L}$.

Definición 4.3.8 *G^t_α es el grafo triangulado del grafo $G = (\mathbf{X}, \mathbf{L})$ según el orden α si $G^t_\alpha = (\mathbf{X}, \mathbf{L}')$, donde $\mathbf{L}'$ esta compuesto por todas las aristas de $\mathbf{L}$ más aquellas que se han añadido durante el proceso de eliminación de nodos según el orden α.*

Definición 4.3.9 *Definiremos el peso del grafo triangulado G^t_α como:*

$w(G^t_\alpha) = log_2 \sum_{C_i \in C} \prod_{V_j \in C_i} r_j$, *donde C es el conjunto de conglomerados de G^t_α.*

Por tanto, trataremos en el EDA de minimizar la función de evaluación $w(G^t_\alpha)$, donde el individuo es el propio α.

4.3.2 Parametrización del EDA

Hemos empleado dos algoritmos EDA descritos en el capítulo 3, el UMDA (107), que asume independencia total entre todas las variables, y MIMIC (23), que asume dependencias entre pares de variables, ambos en dominios tanto discretos como continuos. La representación de individuos continuos y discretos asociados a un orden también se ha descrito en el mismo capítulo, así como las redes Sparse y Dense utilizadas en los experimentos. La generación inicial se obtiene aleatoriamente.

4.3.3 Función de evaluación del individuo

Como hemos comentado antes, tratamos de minimizar el peso del grafo triangulado. La función recibirá como parámetro el individuo, que no será otra cosa que un orden de borrado de variables para realizar la triangulación (esto es, partiremos del grafo no dirigido y ya moralizado).

4.3.4 Planteamiento de los experimentos

Hemos lanzado los experimentos para una población de $M = 10, 50, 100, 150, 250$ y 500 individuos para las dos redes. Se han realizado 50 experimentos por población y red, esto es, 50 lanzamientos de cada EDA: UMDA para individuos continuos ($UMDA_c$), UMDA para individuos discretos ($UMDA_d$), MIMIC para individuos continuos ($MIMIC_c$) y MIMIC para individuos discretos ($MIMIC_d$). Cada lanzamiento está parametrizado de la siguiente manera:

- Muestreo - Cada M individuos nuevos evaluados, se imprime el mejor orden encontrado junto con el valor de su evaluación.
- Criterio de parada - Cuando se llega a las 100 muestras, el algoritmo finaliza. En ocasiones la parada no es exacta, ya que hay que dar tiempo a que finalize la generación en curso, y a veces se repiten los órdenes propuestos por el EDA. No hay un criterio de convergencia. Esto se hace así para poder comparar los resultados con un experimento aleatorio en el que también se evaluan $M \cdot 100$ individuos, de manera que podamos comprobar si el uso del EDA realmente tiene sentido y propone órdenes mejores que uno aleatorio cualquiera. A veces la repetición de órdenes ya propuestos previamente es muy intensa porque el algoritmo converge: esto puede hacer que no se produzcan muestras en mucho tiempo. Para evitar esta situación, si el EDA propone M órdenes ya visitados de manera consecutiva, se imprime una muestra. En otras pruebas no presentadas en este trabajo con periodos de evolución más largos (mayor número de generaciones), no se ha observado una mejora sustancial.
- Grupo de selección - Se seleccionan siempre los $M/10$ mejores individuos de la población para generar la red Bayesiana asociada al EDA, esto es, los asociados a una triangulación de menor peso, excepto en el caso de $M = 10$, donde seleccionaremos los 5 mejores. Veremos en el capítulo 7 que esta elección puede condicionar bastante los resultados.
- Nueva población - Sólo se mantienen los cinco mejores individuos de la población anterior, generándose el resto.
- Otros parámetros - No se utiliza elitismo y la selección de los individuos se realiza por truncación.

La máquina utilizada para realizar todos los experimentos (esto es, para todos los tamaños de población, tipos de EDA y redes) ha sido un Alpha Server GS1280

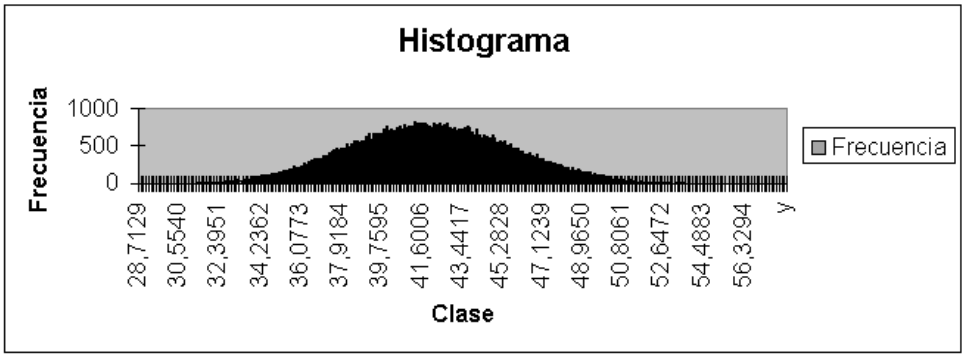

Figura 4.6: Distribución de la función de evaluación para la red Sparse de 10^6 órdenes aleatorios. Evaluación del mejor orden encontrado: 26.63. Evaluación del peor: 59.57.

con microprocesadores Alpha a 1.4 Ghz y 10 Gb de RAM. Esta máquina, de las más potentes para procesamiento serial, tardaba alrededor de cuatro días en realizar todas las pruebas.

4.4 Resultados obtenidos

En la Figura 4.6 podemos ver la distribución de los valores de la función de evaluación para 10^6 órdenes aleatorios. Esta figura nos sirve para hacernos una idea de la distribución de valores de la función en el espacio de búsqueda, que como vemos sigue una campana de Gauss muy estirada.

En los Cuadros 4.5 y 4.6 podemos ver los mejores resultados medios de las 50 ejecuciones y, respectivamente, los EDAs continuos y discretos para la red Sparse. En los Cuadros 4.8 y 4.9 tenemos la misma información, pero para la red Dense. En el Cuadro 4.10 se pueden ver los resultados para las dos redes del experimento aleatorio. En el Cuadro 4.11 podemos ver qué tipo de EDA (MIMIC o UMDA) ha conseguido los mejores resultados medios, tanto para la red Sparse como para la red Dense y todos los tipos de población. En las Figuras 4.7 y 4.8 vemos una comparación para poblaciones de 250 individuos de todos los tipos de EDA propuestos.

Por otra parte, en las Figuras 4.9 y 4.10 podemos ver una comparación por poblaciones del comportamiento de todos los EDAs para las dos redes. En la Figura 4.11 aparece la misma información para los experimentos aleatorios.

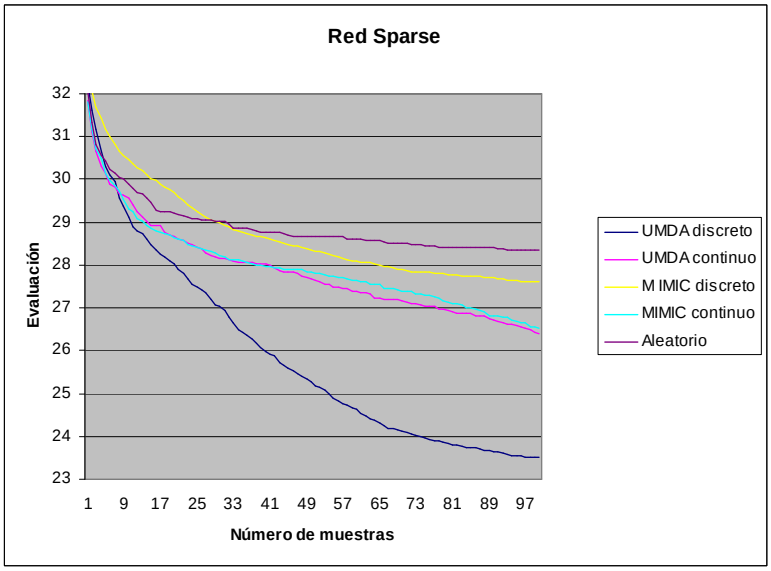

Figura 4.7: Comparación para Sparse de los EDA continuos y discretos para poblaciones de 250 individuos.

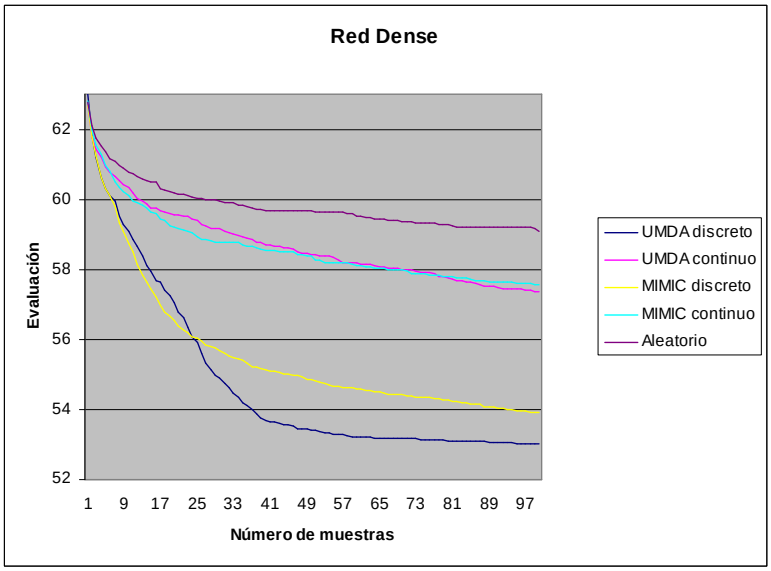

Figura 4.8: Comparación para Dense de los EDA continuos y discretos para poblaciones de 250 individuos.

Cuadro 4.5: Resultados medios de 50 ejecuciones para Sparse y EDAs continuos.

	MIMIC_c	UMDA_c	Mejor MIMIC_c	Mejor UMDA_c
M=10	29,52±2,40	29,82±2,08	26,34	26,53
M=50	27,78±2,24	27,54±1,53	24,59	25,11
M=100	26,79±2,35	26,80±2,34	23,48	23,48
M=150	26,34±2,31	26,91±1,79	23,52	23,41
M=250	26,53±1,41	26,41±1,17	23,82	23,89
M=500	27,55±0,23	27,43±0,33	26,65	26,26

Cuadro 4.6: Resultados medios de 50 ejecuciones para Sparse y EDAs discretos.

	MIMIC_d	UMDA_d	Mejor MIMIC_d	Mejor UMDA_d
M=10	30,52±1,34	30,03±1,51	27,11	27,12
M=50	29,16±1,40	28,34±0,95	25,46	26,32
M=100	27,61±1,61	26,55±0,90	24,20	24,38
M=150	27,58±0,58	25,01±1,06	25,63	23,57
M=250	27,60±0,89	23,51±0,16	25,44	22,67
M=500	27,72±0,63	23,35±0,09	25,73	22,66

4.5 Conclusiones

Podemos extraer las siguientes conclusiones de los resultados mostrados en la anterior sección:

- Según el Cuadro 4.11, UMDA parece superar a MIMIC en general, sobre todo

Cuadro 4.7: Repetición de los experimentos para MIMIC_d con $N = 10$.

	MIMIC_d	Mejor MIMIC_d
M=100	27,02±1,42	24,01
M=150	27,15±1,36	23,82
M=250	26,80±1,59	23,91
M=500	26,49±1,24	23,99

Cuadro 4.8: Resultados medios de 50 ejecuciones para Dense y EDAs continuos.

	MIMIC$_c$	UMDA$_c$	Mejor MIMIC$_c$	Mejor UMDA$_c$
M=10	60,24±1,52	60,09±1,79	56,08	56,11
M=50	58,64±1,45	58,68±2,16	55,23	54,78
M=100	57,87±0,95	58,01±1,23	55,63	55,26
M=150	57,81±1,11	57,90±1,04	53,40	54,91
M=250	57,56±0,82	57,37±1,08	54,96	54,93
M=500	57,92±0,49	57,97±0,31	56,19	56,51

Cuadro 4.9: Resultados medios de 50 ejecuciones para Dense y EDAs discretos.

	MIMIC$_d$	UMDA$_d$	Mejor MIMIC$_d$	Mejor UMDA$_d$
M=10	58,02±2,34	59,08±1,39	55,08	56,55
M=50	55,87±3,72	56,74±1,89	52,88	53,97
M=100	54,50±0,92	53,54±1,47	52,00	51,23
M=150	54,61±0,96	52,89±1,36	52,78	51,09
M=250	53,90±0,87	53,00±1,37	51,66	51,13
M=500	54,26±1,02	52,58±1,15	52,32	51,11

Cuadro 4.10: Resultados medios y mejores de 50 ejecuciones aleatorias para Sparse y Dense.

	Media Sparse	Mejor Sparse	Media Dense	Mejor Dense
M=10	30,51±0,99	28,41	61,40±1,13	57,24
M=50	29,34±0,53	27,40	59,87±0,84	58,08
M=100	28,84±0,73	26,30	59,68±0,48	58,04
M=150	28,43±0,41	26,26	59,40±0,76	57,12
M=250	28,34±0,29	27,07	59,09±0,77	56,30
M=500	27,56±0,45	26,11	58,50±0,62	56,04

Cuadro 4.11: Mejor EDA para cada combinación de tamaño de población y tipo de variable.

	Continuo Sparse	Discreto Sparse	Continuo Dense	Discreto Dense
M=10	MIMIC	UMDA	UMDA	MIMIC
	29,52 < 29,82	30,03 < 30,52	60,09 < 60,24	58,02 < 59,08
M=50	UMDA	UMDA	MIMIC	MIMIC
	27,54 < 27,78	28,34 < 29,16	58,64 < 58,68	55,87 < 56,74
M=100	MIMIC	UMDA	MIMIC	UMDA
	26,79 < 26,80	26,55 < 27,61	57,87 < 58,01	53,54 < 54,50
M=150	MIMIC	UMDA	MIMIC	UMDA
	26,34 < 26,91	25,01 < 27,58	57,81 < 57,90	52,89 < 54,61
M=250	UMDA	UMDA	UMDA	UMDA
	26,41 < 26,53	23,51 < 27,60	57,37 < 57,56	53,00 < 53,90
M=500	UMDA	UMDA	MIMIC	UMDA
	27,43 < 27,55	23,35 < 27,72	57,92 < 57,97	52,58 < 54,26

en el dominio discreto (es el mejor EDA en 15 de 24 ocasiones). Las diferencias pueden llegar a ser grandes. En concreto, llama la atención el resultado para la red Sparse en el dominio discreto, donde el mejor resultado para MIMIC_d no es capaz de bajar, en poblaciones de 500 individuos, de $25,73$, y el resultado medio es incluso peor que el del experimento aleatorio. Pero basta con lanzar una ejecución para $N = 10$ (ver cuadro 4.7) para darnos cuenta de lo dependiente que es el resultado de alguno de los parámetros del EDA. En el dominio continuo las diferencias no son tan acusadas. No resulta difícil elegir como mejor a un EDA concreto: el UMDA discreto con tamaños de población grandes obtiene buenos resultados, tanto para la red Sparse como para Dense. Las dispersiones también son similares en todos los casos.

- La comparación directa entre los dominios continuos y discretos (ver Figuras 4.7 y 4.8) muestran resultados diferentes según el tipo de aprendizaje, como queda de manifiesto en las figuras 4.12 y 4.13. En el test de suma de rangos de Wilcoxon para comparar medianas, UMDA_d con poblaciones de 500 individuos es mejor que cualquier otro EDA con un $p-$valor de prácticamente 0. No hay más diferencias estadísticamente significativas, exceptuando la compa-

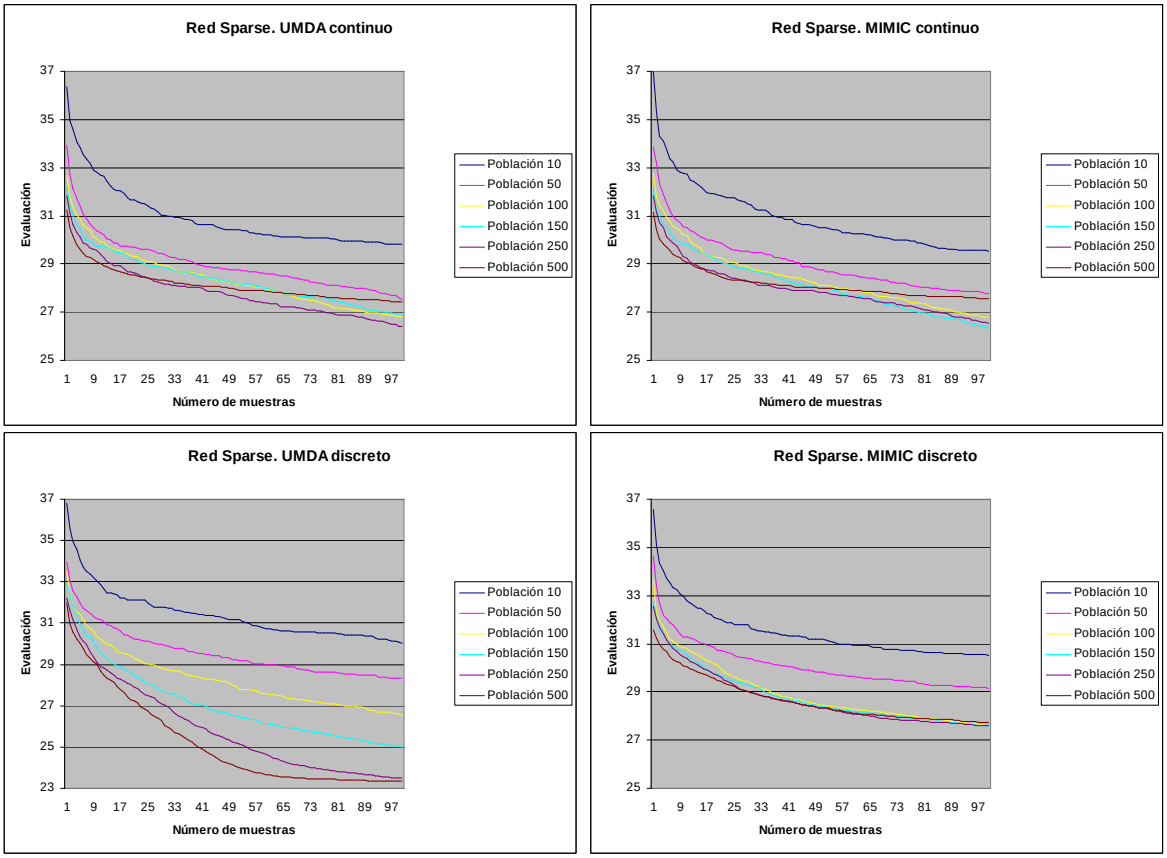

Figura 4.9: Comparación para Sparse de los EDA continuos y discretos por poblaciones.

ración entre MIMIC$_d$ y UMDA$_c$ (0,04 a favor del segundo). Con poblaciones de 250 individuos se confirman los resultados anteriores, empeorando aún más el comportamiento de MIMIC$_d$, ahora también con diferencia estadísticamente significativa con MIMIC$_c$ (0,00001). En el caso de Dense y poblaciones de 500 individuos (ver figura 4.13), se repiten las mismas diferencias para UMDA$_d$: este EDA resulta ser el mejor con un $p-$valor de 0 en el caso de MIMIC$_c$ y UMDA$_c$, y con un $p-$valor de $2,08 \cdot 10^{-10}$ en el caso de MIMIC$_d$. Sin embargo, esta vez MIMIC$_d$ es mejor que MIMIC$_c$ y que UMDA$_c$ con un $p-$valor de 0. Las diferencias estadśticamente significativas se repiten en las poblaciones de 250 individuos.

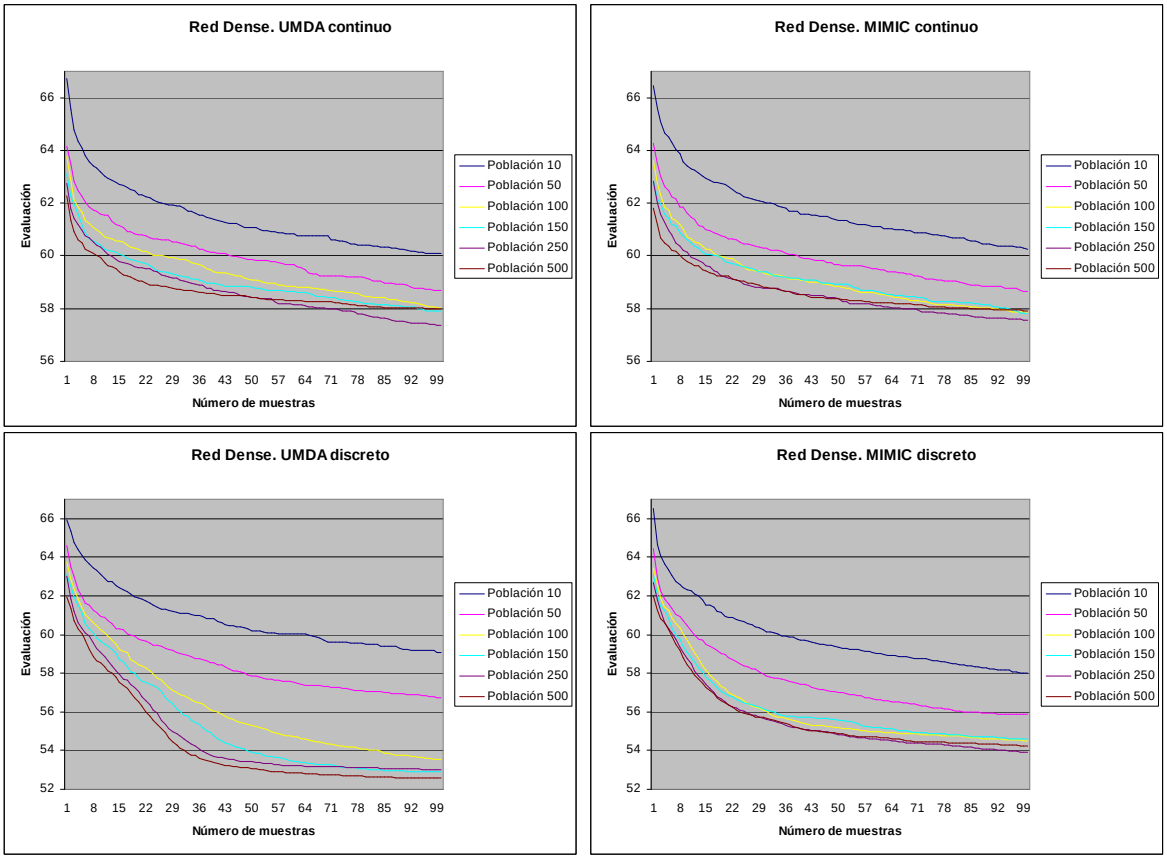

Figura 4.10: Comparación para Dense de los EDA continuos y discretos por poblaciones.

- En cuanto al tamaño de las poblaciones (ver las Figuras 4.9 y 4.10), mientras que en los EDA discretos y UMDA las poblaciones grandes obtienen mejores resultados, en MIMIC y en los EDA continuos en general esta tendencia no se mantiene. En los experimentos aleatorios (ver Figura 4.11) la mejora con el tamaño de la población es más constante debido simplemente al mayor número de individuos diferentes evaluados.

- Las comparaciones con otros métodos ponen de manifiesto un comportamiento razonable de un EDA concreto, UMDA$_d$. La mejor media para Sparse es de $23,35$, mientras que la *peor* para el enfriamiento estadístico en los experimentos llevados a cabo por Kjaerulff en (85) resulta ser de $23,89$. En (94), Larrañaga

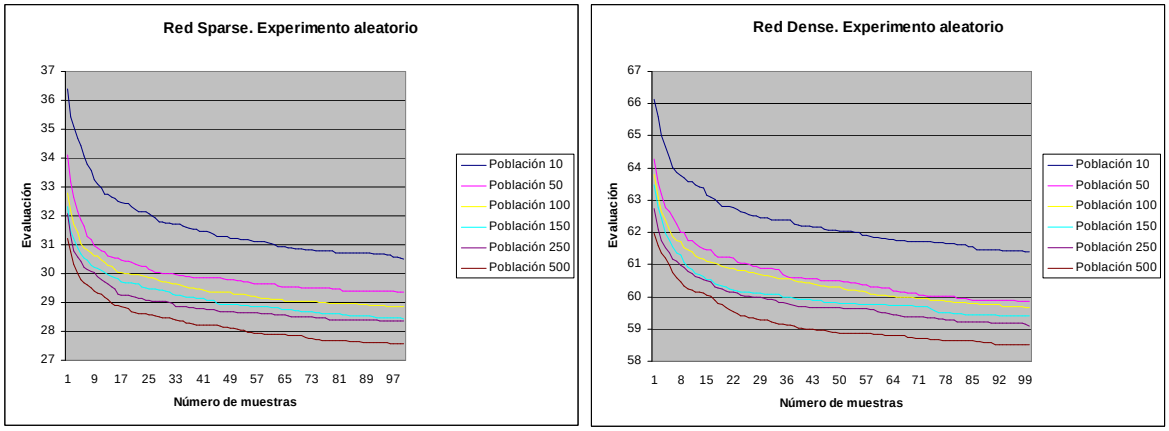

Figura 4.11: Comparación para Sparse y Dense de los experimentos aleatorios por poblaciones.

y col. obtienen con los algoritmos genéticos medias inferiores a $23,05$ en 334 de las 1296 combinaciones de parámetros para estos algoritmos utilizadas en los experimentos. En cualquier caso, el comportamiento del resto de los EDAs utilizados es bastante malo.

En general, podemos decir que los resultados con la triangulación son satisfactorios únicamente para un EDA: UMDA$_d$. Como veremos en el capítulo 7, y como queda de manifiesto en éste, una mala elección de los parámetros del EDA (tipo de aprendizaje, tamaño de la población, tamaño del grupo de selección, etc) para un problema concreto puede desembocar en unos malos resultados (vease el ejemplo para MIMIC$_d$ y la red Sparse de los cuadros 4.6 y 4.7).

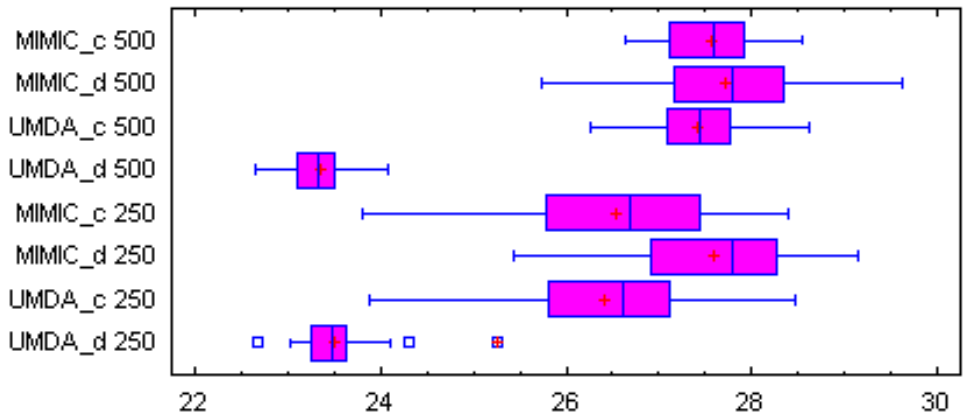

Figura 4.12: Gráfico de cajas y bigotes para comparar todos los EDAs aplicados a la red Sparse para poblaciones de 250 y 500 individuos. Los datos están divididos en cuatro áreas de frecuencia iguales (cuartiles). La caja engloba el 50 % central y la mediana aparece dibujada como una línea dentro de la caja.

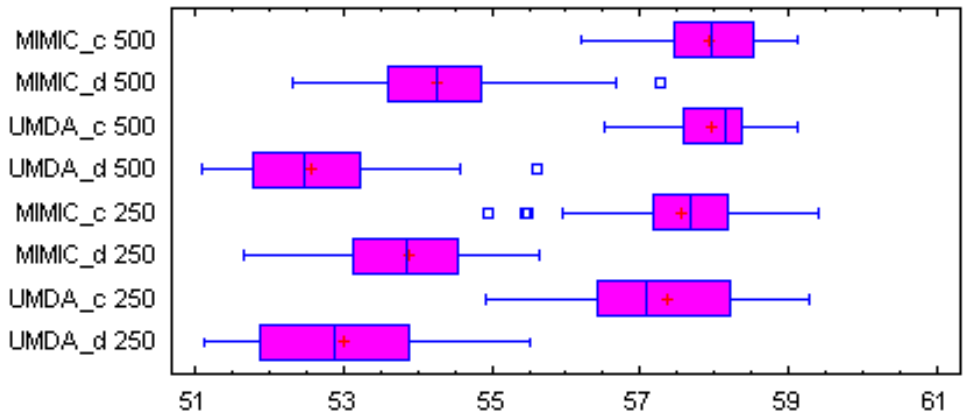

Figura 4.13: Gráfico de cajas y bigotes para comparar todos los EDAs aplicados a la red Dense para poblaciones de 250 y 500 individuos. Ver Figura 4.12 para una descripción más detallada del gráfico.

Capítulo 5

Órdenes Óptimos en el Aprendizaje Estructural

'Lo que no se parece a nada no existe.'

Paul Valéry

La búsqueda del orden óptimo de un conjunto de variables para resolver un problema computacional es un subproblema con el que nos podemos encontrar en circunstancias muy diversas. Una de esas situaciones aparece cuando tratamos de aprender una red Bayesiana a partir de un conjunto de datos. La búsqueda en el espacio de posibles estructuras para encontrar la que más verosímilmente puede dar lugar al conjunto de datos propuesto mediante métodos exactos o de fuerza bruta es a menudo impracticable, sobretodo cuando el numero de variables es grande. Una aproximación heurística muy popular es el algoritmo de Cooper y Herskovits, también llamado K2. Pero la bondad del comportamiento de este algoritmo depende fuertemente de un orden previo de las variables que ha de estar prefijado.

Este capítulo, que recoge el trabajo publicado por Romero y col. en (128), expondrá los resultados de la aplicación de los EDAs a la búsqueda del mejor orden, esto es, aquel que de lugar, tras la aplicación del algoritmo K2, a la red Bayesiana que más verosímilmente explique un conjunto de datos [1]. Dividiremos el capítulo en las siguientes secciones:

- En la Sección 5.1 se expone brevemente en que consiste el algoritmo K2 y cómo

[1] En realidad, el algoritmo K2 utiliza ya la verosimilitud marginal como métrica, como veremos más adelante.

podemos estimar lo cerca o lejos que está una red Bayesiana dada de explicar el conjunto de datos propuesto.

- En la Sección 5.2 se exponen otros problemas en los que se ha utilizado la misma táctica de búsqueda en el espacio de órdenes en lugar de búsquedas en espacios más grandes.
- En la Sección 5.3 se proponen varios tipos de EDA para resolver el problema.
- En la Sección 5.4 se presentan los resultados obtenidos con los EDA y se comparan con los de otros métodos.
- Finalmente, en la Sección 5.5 se exponen las principales conclusiones a las que los resultados obtenidos dan lugar.

5.1 El algoritmo K2

Sea D un conjunto de datos extraido de algún tipo de medición real que contiene N ejemplos obtenidos de un dominio $\mathbf{X} = (X_1, ..., X_n)$ con n variables. En lugar de obtener el modelo gráfico probabilístico (MGP), esto es, la estructura de la red y las probabilidades condicionadas entre las variables que forman los nodos de la red, a menudo el trabajo se limita a obtener la estructura S, por ejemplo cuando lo único que nos interesa es descubrir las relaciones de dependencia e independencia de las variables. En la aplicación de los EDA al problema nos centraremos en la obtención de la estructura.

El espacio de estructuras crece exponencialmente con el número de nodos, lo cual imposibilita la búsqueda sistemática para redes grandes. Robinson propone en (126) la siguiente función para calcular el número Ξ de posibles estructuras para n nodos:

$$\Xi(n) = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} 2^{i(n-i)} \Xi(n-i) \tag{5.1}$$

Para $n = 10$, tenemos $4,2 \times 10^{18}$ posibles estructuras, que resultan imposibles de visitar una a una. El aprendizaje estructural es generalmente un problema NP-duro (42). Una de las tácticas que podemos utilizar para reducir el espacio de búsqueda, que es ya un procedimiento estándar, es considerar solo aquellas estructuras que, dado un orden α, se cumple que $X_i \in \Pi_j \Rightarrow i <_\alpha j$, siendo X_i una variable de la estructura y Π_j el conjunto de padres de X_j. Al reducir de esta manera el espacio,

estamos, sin embargo, obteniendo estructuras que dependen mucho del método de búsqueda, y en determinados dominios no podemos saber a priori la probabilidad de una estructura dada. Esta dependencia desaparece si realizamos una búsqueda ulterior en el espacio de órdenes. Otras tácticas, más artificiales, consisten en limitar el número de padres que puede tener un nodo, o rechazar modelos muy complejos.

La ventaja que nos da el trabajar con órdenes en vez de directamente con estructuras aparece en varios trabajos. Además de que el espacio de búsqueda es, evidentemente, menor ($n!$ en lugar de $2^{O(n^2)}$, donde n es el número de nodos), la convergencia de la solución es mas rápida, y ésta es más suave (63). Por tanto, podemos afrontar la búsqueda sin necesidad de añadir más restricciones.

Sin embargo, si el número de nodos se incrementa lo suficiente, $n!$ se convierte en un número muy grande y los problemas prácticos vuelven a aparecer. Por tanto, vamos a necesitar aplicar heurísticos no solo al problema de encontrar una estructura óptima para el conjunto de datos D con un orden de las variables previo (esto es, la búsqueda en el espacio de las estructuras restringido a aquellas que cumplen la condición descrita en el primer párrafo de esta sección), sino que también habremos de aplicar otro heurístico para la búsqueda en el espacio de los órdenes. Llamaremos al primer heurístico el *Paradigma de Búsqueda de la Estructura* (PBE), y al segundo el *Paradigma de Búsqueda del Orden* (PBO). En este capítulo utilizaremos como PBE un algoritmo voraz (el citado algoritmo K2 (45)), y como PBO pondremos a prueba el *univariate marginal distribution algorithm* (UMDA) (107) y el *mutual information maximization for input clustering* (MIMIC) (23), ya descritos en el capítulo 2, ambos tanto en dominios discretos como en continuos.

5.1.1 Coste de la estructura obtenida a partir del orden

El algoritmo K2 propuesto por Cooper y Herskovits (45) se basa en una función de coste para una estructura concreta que se utiliza para evaluar la calidad de un orden:

$$P(B_S|D) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{5.2}$$

donde q_i es el número de posibles instanciaciones de $\mathbf{\Pi}_i$ en D, N_{ijk} el número de casos en D en los cuales X_i está instanciada como su $k-$ésimo posible valor y π_i tiene su $j-$ésima posible instanciación, r_i el número de valores posibles de X_i, y siendo $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$. Esto es cierto si y solo si:

- Los casos ocurren independientemente unos de otros, esto es, ningún caso o conjunto de casos tiene alguna influencia en el resto.
- Dada la estructura, la densidad de los parametros es uniforme.
- El conjunto de datos es completo, esto es, no tenemos datos desconocidos, sino que todas las variables tienen un valor asignado en cada caso.

De (5.2) podemos asociar un coste a una variable X_i utilizando la función:

$$g(i, \mathbf{\Pi}_i) = \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{5.3}$$

5.1.2 Descripción del algoritmo

El algoritmo K2 es un algoritmo voraz que busca para cada nodo X_i el conjunto de padres que maximiza la función $g(i, \mathbf{\Pi}_i)$, añadiendo un padre en cada paso hasta que la adición de un padre no mejora el valor de $g(i, \mathbf{\Pi}_i)$. Podemos ver una descripción del algoritmo en la Figura 5.1.

5.2 Búsquedas en espacios de órdenes

Se han propuesto muchos métodos de generación de estructuras a partir de bases de datos de casos o muestras. Muchos de ellos se limitan a buscar en el espacio de los posibles grafos o clases de equivalencia, basándose en tests de independencia condicional o en la optimización de una métrica concreta. Como hemos comentado al comienzo del capítulo, también podemos buscar en el espacio de estructuras estableciendo un orden previo de las variables que determina la búsqueda misma, que es lo que ocurre con el algoritmo K2, y realizar una búsqueda ulterior sobre el espacio de los órdenes. Podemos encontrar varias aplicaciones de esta estrategia a la obtención de estructuras en la literatura especializada. En lo que sigue denominaremos B_S a la estructura del modelo gráfico probabilistico y B_p a θ_S, esto es, al vector de probabilidades condicionadas asociadas a la estructura B_S, para poner de manifiesto que el modelo es, en este caso, una red Bayesiana.

En (30), Bouckaert propone también el K2 como algoritmo de búsqueda estructural y, como PBO, un algoritmo que manipula un orden dado α de las variables con operaciones similares al reverso de arcos. Sólo se ejecutan aquellas operaciones

```
Procedimiento K2 (X, D, α, MaxPadres, L)
  L ← ∅
  Para i de 1 a n Hacer
    Π_i ← ∅
    G_viejo ← g(i, Π_i)
    Seguir ← Cierto
    Mientras SEGUIR y |Π_i| < MaxPadres Hacer
      Escoger el nodo X_j de X t. q. j <_α i, X_j ∉ Π_i y
                                    g(i, Π_i ∪ {X_j} es máximo
      G_nuevo ← g(i, Π_i ∪ {X_j}
      Si G_nuevo > G_viejo entonces
        G_viejo ← G_nuevo
        Π_i ← Π_i ∪ {X_j}
      SiNo
        SEGUIR ← Falso
      FinSi
    FinMientras
  FinPara
  Para i de 1 a n Hacer
    Incorporar todas las aristas implicitas en Π_i a L
  FinPara
```

Figura 5.1: Algoritmo K2 para la obtención de una red Bayesiana para un conjunto de nodos $\mathbf{X}$, una ordenación de los mismos α y una base de datos de casos D. El algoritmo devuelve la estructura de la red Bayesiana en L. La variable $MaxPadres$ es una limitación en el número de padres que puede tener cada nodo.

que dan lugar a un grafo acíclico dirigido que representa el mismo modelo de independencia presente en la estructura antes de su aplicación, incrementándose en cada paso el tamaño del conjunto de independencias.

En (91), Larrañaga y col. usan los algoritmos genéticos (106) como PBO. Para generar el grafo acíclico dirigido, también utilizan el K2 de Cooper y Herskovits (45), esto es, también se utiliza la métrica comentada en la sección anterior. El algoritmo genético utilizado está basado en el GENITOR, desarrollado por Whitley (153). Este algoritmo combina varias operaciones genéticas de cruce y mutación para ir obteniendo distintos órdenes, esto es, el dato de entrada que necesita el K2 para

empezar a buscar la estructura más adecuada.

Para Friedman y Koller (63), el objetivo principal es computar $P(B_S|D) \propto P(D|B_S)P(B_S)$ para buscar en el espacio de las estructuras. $P(D|B_S)$, la *probabilidad marginal* de los datos dada una estructura, se obtiene integrando todos los posibles parámetros de la red Bayesiana. Esto, para redes grandes, es imposible, por lo que los autores reducen la complejidad del problema limitando el espacio de las estructuras a aquellas cuyos nodos sólo pueden tener como máximo k padres, donde k es una constante, y además restringen el conjunto de posibles padres de un nodo con un orden α concreto de las variables (en el sentido que ya se ha explicado anteriormente). Además, utilizan toda una bateria de argucias computacionales, como reducir aún más el conjunto de posibles padres de un nodo y el de posibles familias a las que pueden dar mejores puntuaciones mediante una pre-computación. Con todas estas restricciones, la estimación de $P(B_S|D, \alpha)$ es finalmente posible. Para buscar en el espacio de órdenes, los autores construyen una cadena de Markov homogénea en la que el espacio de estados son los $n!$ posibles órdenes de las variables. En este caso, la construcción de la cadena de Markov utilizando un algoritmo de muestreo Metrópolis estándar necesita garantizar que la distribución invariante sobre los estados es $P(\alpha|D)$.

Acid y col. (2) usan el algoritmo BENEDICT-*step* como PBE. Este algoritmo está basado en una metodología híbrida que combina tests de independencia y métricas. La métrica mide la discrepancia entre las listas de $d-$separación de una red candidata B_S y las que aparecen en D. Para ello, se usa la entropía cruzada de Kullback-Leibler. Como siempre, en cuanto el número de nodos de B_S se incrementa, comienzan a aparecer los problemas computaciones para calcular las listas de $d-$separación, ya que su complejidad y su tamaño crece exponencialmente. Por ello, se restringe de nuevo el tamaño del espacio de las posibles soluciones teniendo en cuenta únicamente las independencias condicionales para cada par de variables no adyacentes X_i y X_j, dado el conjunto de tamaño mínimo $D_s(X_i, X_j)$ que las $d-$separa (1). Estas listas de $d-$separación son representativas de todas las $d-$separaciones que aparecen en B_S. El método de búsqueda que usa la métrica es un algoritmo voraz simple que toma en cuenta, como medida de discrepancia global, la suma de las discrepancias locales, y añade un arco solamente si la discrepancia global resultante disminuye. Esta adición respeta un orden dado. El orden óptimo se busca en este caso con enfriamiento estadístico (83).

En (36), de Campos y Puerta utilizan tambien la métrica K2 como método de puntuación para evaluar la calidad de una estructura. Para obtener ésta, utilizan un algoritmo simple de tipo ascenso por la colina para buscar el conjunto de padres más

idóneo para cada nodo utilizando operadores de adición y borrado de arcos. Para buscar dentro del espacio de órdenes, los autores utilizan la búsqueda de la vecindad variable (*variable neighbourhood search*) (73) basado en órdenes. Primero, seleccionan un conjunto de órdenes de vecindad N_k (con $k = 1, \ldots, kmax$), donde $N_1(\alpha)$ es la vecindad del orden α que resulta del intercambio de dos posiciones cualesquiera y, en general:

$$\alpha'' \in N_k(\alpha) \Leftrightarrow \alpha'' \in N_1(\alpha') \wedge \alpha' \in N_{k-1}(\alpha) \tag{5.4}$$

Después, se busca en el espacio de órdenes de vecindad (primero, en N_k, y si no hay mejora, en N_{k+1}). Después de seleccionar un orden aleatorio de N_k, se usa una búsqueda de tipo ascenso por la colina para buscar en el espacio de órdenes con un operador de intercambio de dos posiciones. Para acelerar el proceso, establecen un número r que limita la distancia entre variables a ser intercambiadas en un orden.

La búsqueda en el espacio de órdenes propuesto por Puerta (122) combina el algoritmo K2SN con algoritmos de colonias de hormigas (55). K2SN es una modificación del algoritmo K2 que no necesita de un orden previo de las variables, ya que el K2SN mismo determina dicho orden. En casa paso, K2SN usa el algoritmo K2 para descubrir el nodo y el conjunto de padres que minimiza la métrica K2, añadiendo el nodo a la red y por lo tanto, estableciendo su posición. En el heurístico denominado colonia de hormigas, la elección del nodo no se basa solo en la métrica K2, sino tambien en una cantidad de *feromonas*. Cuanta más feromona está asociada a un determinado arco, más probabilidades hay de seleccionar dicho arco. Cada vez que el arco es seleccionado por una hormiga (lo cual equivale a una ejecución de hormiga + algoritmo K2SN), su cantidad de feromona asociada aumenta. Finalmente, hay una búsqueda ulterior en el espacio de cada órden propuesto por la colonia de hormigas mediante un algoritmo de ascenso por la colina que se describe en (36).

En el Cuadro 5.1 podemos ver un pequeño resumen de esta sección.

5.3 Solución del problema con EDAs

En nuestro caso, cada individuo-orden propuesto por el EDA será utilizado como entrada del algoritmo K2, y el valor de la función g será la evaluación de cada individuo. Formalicemos brevemente a continuación el problema, en el que tratamos de maximizar $P(B_S|D)$, donde B_S es la estructura de una red Bayesiana. La demostración de todos los teoremas puede verse en (45).

Cuadro 5.1: Resumen de los trabajos de aprendizaje estructural en el espacio de órdenes para el problema del aprendizaje estructural.

Autores	PBE + PBO
Bouckaert (1992)	K2 + conservación de independencias
Larrañaga y col (1996)	K2 + algoritmos genéticos
Friedman, Koller (2000)	restricción del espacio + MCMC
Acid y col (2001)	BENEDICT-step + enfriamiento estadístico
de Campos, Puerta (2001)	ascenso por la colina + VNS basado en órdenes
Puerta (2001)	ascenso por la colina + K2SN y colonias de hormigas

Teorema 5.3.1 *Sea $X = \{X_1, \ldots, X_n\}$ un conjunto de n variables, B_S la estructura de una red Bayesiana, D una base de datos de casos y $P(B_S|D)$ la probabilidad a posteriori de que B_S explique D. Entonces, se da el hecho de que $P(B_S|D) \propto P(B_S, D)$, y buscar la estructura B_S que maximiza $P(B_S|D)$ es equivalente a buscar la que maximiza $P(B_S, D)$.*

Teorema 5.3.2 *Si todas las variables X_i para $i = 1 \ldots n$ del conjunto de variables X son discretas, podemos expresar la probabilidad $P(B_S|D)$ como:*

$$P(B_S|D) = \int_{B_p} P(D|B_S, B_p) f(B_p|B_S) P(B_S) dB_p \tag{5.5}$$

donde B_p es un vector en el que los valores son las asignaciones de probabilidad condicionada asociadas a la estructura B_S y f es la función de densidad condicional de B_p dado S.

Teorema 5.3.3 *Si los casos que aparecen en D son independientes unos de otros, entonces podemos expresar la masa de probabilidad $P(D|B_S, B_p)$ como:*

$$P(D|B_S, B_p) = \prod_{i=1}^{m} P(C_i|B_S, B_p) \tag{5.6}$$

donde m es el número de casos de D y C_i es el i-ésimo caso de D.

Teorema 5.3.4 *Si además de las condiciones de los teoremas anteriores, no hay casos en los que haya variables que no tengan un valor asignado y además la función de densidad $f(B_p|B_S)$ es uniforme (esto es, que no hay vectores de probabilidad B_p que tengan a priori una probabilidad mayor o menor dada la estructura B_S, sin tener en cuenta a D, sino que todos los vectores son igualmente probables), entonces $P(B_S, D)$ puede ser escrita como:*

$$P(B_S, D) = P(B_S) \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{5.7}$$

donde r_i es el número de posibles valores de X_i y N_{ijk} es el número de casos en D en los que la variable X_i tiene su valor k-ésimo y su conjunto de padres $\mathbf{\Pi_i}$ esta instanciado en su j-ésimo posible valor. N_{ij} se define como $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

Teorema 5.3.5 *Si además las condiciones de los teoremas anteriores, aplicamos un orden α a X de manera que $X_i \in \Pi_j \Rightarrow i <_\alpha j$ y asumimos que cualquier estructura B_S tiene a priori la misma probabilidad, la ecuación 5.7 puede ser rescrita como:*

$$P(B_S, D) = c \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{5.8}$$

donde c es la probabilidad a priori de cualquier estructura B_S.

Por tanto, maximizar $P(B_S|D)$, que es nuestro objetivo en el aprendizaje de una red a partir de una base de datos de casos, dadas todas las condiciones descritas en los teoremas, es equivalente a maximizar la expresión:

$$c \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{5.9}$$

teniendo en cuenta que $\mathbf{\Pi_i}$ respeta en todo momento el orden α que hemos impuesto a X. Si tendemos a maximizar en todo momento la expresión:

$$\prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \prod_{k=1}^{r_i} N_{ijk}! \tag{5.10}$$

conseguiremos también maximizar $P(B_S, D)$: de ahí el valor de la función $g(i, \mathbf{\Pi}_i)$ y de la métrica K2 en general, que es, como hemos dicho, la que utilizaremos para evaluar un individuo del EDA. Esto es lo que se hace en el algoritmo K2 de la Figura 5.1.

5.3.1 Parametrización del EDA

Hemos empleado los dos algoritmos EDA que hemos aplicado y aplicaremos en todos los problemas afrontados en esta tesis y cuya descripción, como siempre, puede encontrarse en el Capítulo 3 (UMDA (107) y MIMIC (23)), así como la representación de individuos continuos y discretos asociados a un orden. La población inicial se genera, como siempre, aleatoriamente. Cada lanzamiento del EDA está parametrizado de la siguiente manera:

- Criterio de parada - El criterio de parada no está relacionado con ningún criterio de convergencia. Normalmente, el criterio de convergencia, sea cual sea el método, no asegura la terminación del algoritmo. Dado que, como veremos más abajo, nuestros experimentos imprimen una muestra (individuo mejor encontrado hasta entonces y su evaluación) cada M individuos distintos evaluados, hemos hecho que el programa terminara una vez impresas $M \cdot 50$ muestras, de manera que fuera más sencillo comparar los resultados con un experimento aleatorio en el que se generan $M \cdot 50$ órdenes diferentes. Con la red Asia, al dar lugar a un problema más sencillo debido a lo reducido de su tamaño, la repetición en los órdenes propuestos por el EDA es muy alta. Por ello, si se visitan M órdenes consecutivos que ya han sido evaluados en generaciones anteriores, se imprime una muestra y se suma M al contador de individuos distintos evaluados. En el caso de la red Alarm, el algoritmo no puede parar exactamente cuando $M \cdot 50$ individuos han sido evaluados, ya que la generación en curso debe finalizar.

- Grupo de selección - Para seleccionar los individuos usados para estimar la distribución de probabilidad, seleccionamos los $N = \frac{M}{2}$ mejores individuos de la población actual, esto es, los N individuos con mejor métrica K2.

- Nueva población - Sólo se mantiene el mejor individuo de la población anterior, generándose los restantes $M - 1$ individuos en cada paso o generación. Se ha comprobado que mantener en las nuevas poblaciones un alto porcentaje de individuos de las anteriores empeora los resultados, ya que no se deja evolucionar al EDA.

- Otros parámetros - No se utiliza elitismo y la selección de los individuos se realiza por truncación.

5.3.2 Redes utilizadas

El algoritmo K2 utiliza siempre una base de datos de casos a partir de la cual obtiene una estructura que puede estar más o menos cerca de la que hipotéticamente ha generado los datos. Para comprobar lo cerca que está el K2 de devolver la estructura idónea, las bases de datos se han generado a partir de dos redes conocidas, Alarm y Asia, descritas también en el capítulo 2. En el caso de Alarm, hemos generado el fichero de bases de datos mediante el programa Netica, de Norsys Software Corp (46). En el caso de la red Alarm, hemos utilizado el almacén de bases de datos sobre aprendizaje automático de la UCI (111). El número de casos a tener en cuenta ha sido, para ambas redes, de 3000 (en el caso de Alarm, se han usado los 3000 primeros casos de la UCI).

5.3.3 Función de evaluación del individuo

Cada individuo está asociado al orden que utiliza el K2 para encontrar la mejor estructura. La función de evaluación del individuo será pues la métrica del K2 asociada a la mejor estructura encontrada por el algoritmo, dado el orden.

5.3.4 Planteamiento de los experimentos

Hemos lanzado los experimentos para una población de $M = 100$ individuos para Asia y $M = 500$ individuos para Alarm. Se han realizado 10 experimentos por población y red, esto es, 10 lanzamientos de cada EDA (UMDA para individuos continuos ($UMDA_c$), UMDA para individuos discretos ($UMDA_d$), MIMIC para individuos continuos ($MIMIC_c$) y MIMIC para individuos discretos ($MIMIC_d$)). En cuanto al muestreo, cada M individuos nuevos evaluados, se imprime el mejor orden encontrado junto con el valor de su evaluación. Las pruebas se han llevado a cabo en PCs con procesadores Pentium III y IV con distintas cantidades de memoria RAM por debajo de 1Gb, en los que la ejecución de todas las pruebas llevaba aproximadamente una semana.

En cada muestra del mejor individuo de la generación actual, el experimento no sólo imprime el individuo, sino que recoje en la muestra también la siguiente información:

- El valor de la métrica K2 (K2). Es la métrica que intenta maximizar el EDA.

- La distancia de Hamming (DH) entre la estructura de la red a partir de la cual se ha generado el fichero de casos (Asia o Alarm) y el DAG obtenido por el algoritmo K2.

- La distancia del esqueleto (DE), que tiene en cuenta sólo el grafo no dirigido que resulta del DAG generado por el algoritmo K2 y el que resulta de la red original. Cuenta 1 por cada arco que aparece en una red y no lo hace en la otra.

- La distancia mixta (DM), que cuenta 1 por cada arco dirigido que aparece en una red y no en la otra, excepto en el caso de los arcos que aparecen en la otra red pero cambiados de dirección, en cuyo caso cuenta sólo 0,5.

Hay que tener en cuenta que la mejor evaluación (el valor de la métrica K2), no implica necesariamente el mejor valor de las distancias estructurales para una red dada.

5.4 Resultados obtenidos

En las Figuras 5.2 y 5.3 podemos ver, respectivamente, el histograma de la distribución de la métrica K2 para las 8! posibles permutaciones de los nodos de la red Asia así como 20000 órdenes aleatorios, usando en ambas ocasiones el algoritmo de búsqueda K2 para encontrar la estructura. En la Figura 5.4 podemos ver el histograma para 20000 órdenes aleatorios para la red Alarm (el histograma de los 37! posibles órdenes es, evidentemente, imposible de calcular). En ambas figuras podemos apreciar que el problema con Asia es sencillo, ya que la mayor parte de los órdenes devuelve una métrica buena, pero que en el caso de Alarm la búsqueda se antoja más complicada.

Una versión del K2, cuyos resultados no se muestran aquí, cuya diferencia con el algoritmo original consiste en que añade en cada paso los dos padres que mejoran en mayor medida la métrica (en vez de uno), da resultados peores, ya que mueve la mayoría de los órdenes a zonas peores del espacio de búsqueda. Esto sucede probablemente porque el algoritmo mediatiza demasiado la estructura resultante (por ejemplo, cada nodo sólo puede tener un número par de padres). Esto demuestra que el PBE es muy importante a la hora del aprendizaje de estructuras. Hay que tener en cuenta que, en determinados dominios (por ejemplo, en el espacio de estructuras para las redes Asia o Alarm), el modelo de mayor puntuación es varios órdenes de magnitud más apropiado que cualquier otro cuando la cantidad de datos es grande

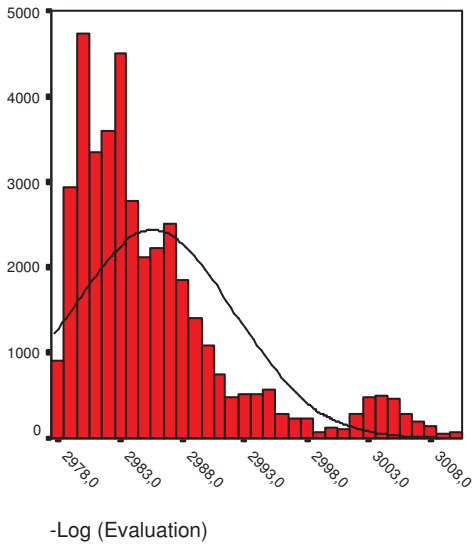

Figura 5.2: Distribución de la métrica K2 para los 8! posibles órdenes de la red Asia en el problema del aprendizaje estructural.

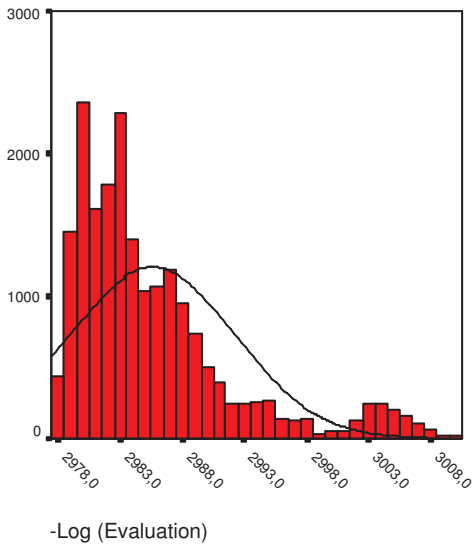

Figura 5.3: Distribución de la métrica K2 para 20000 órdenes aleatorios de la red Asia en el problema del aprendizaje estructural.

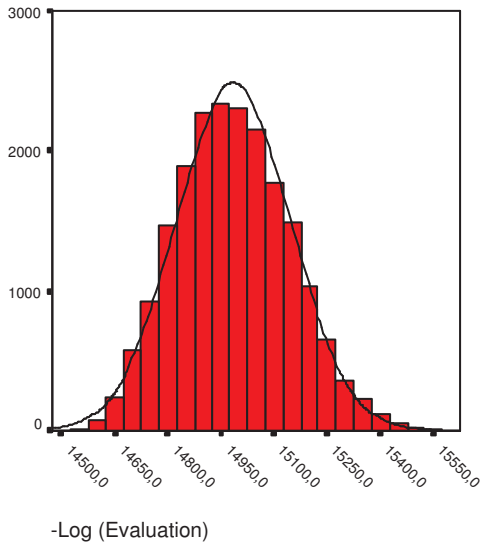

Figura 5.4: Distribución de la métrica K2 para 20000 órdenes aleatorios de la red Alarm en el problema del aprendizaje estructural.

con respecto a su tamaño. Sin embargo, en las búsquedas en espacios de órdenes, hay varias K2-estructuras que pueden explicar bien el conjunto de datos.

En el Cuadro 5.2 podemos ver los mejores resultados medios, junto con sus varianzas, para la red Asia y EDAs continuos (los resultados para los EDAs discretos y el experimento aleatorio resultan ser exactamente los mismos). En los Cuadros 5.3, 5.4 y 5.5 podemos ver los mismos resultados para la red Alarm. Como se puede apreciar, en el caso de Asia el algoritmo K2 es capaz de obtener redes con una puntuación que es incluso mayor que la propia Asia, que es la red a partir de la cual está generado el fichero de casos, mientras que en el caso de Alarm es la estructura de la red original la que devuelve la mejor métrica. Dada la sencillez del problema con Asia, todos los métodos, incluido el experimento aleatorio, pueden devolver el mejor órden posible.

La mejor evaluación media se obtiene con MIMIC$_d$. MIMIC$_d$ parece ser mejor que UMDA$_d$ en dominios discretos, mientras que en los continuos el comportamiento es aproximadamente el mismo.

En el Cuadro 5.6 podemos ver la evaluación del mejor individuo encontrado por cada método en el caso de la red Alarm.

En la Figura 5.5 podemos ver la evolución a lo largo de las 50 muestras de la evaluación media de 10 experimentos para UMDA, MIMIC y el experimento aleatorio

Cuadro 5.2: Medias de 10 ejecuciones para Asia y EDAs continuos en el problema del aprendizaje estructural.

	MIMIC_c	UMDA_c
Evaluación	-2977,52±0,00	-2977,52±0,00
DH	2.00±0,00	2,00±0,00
DE	1.00±0,00	1,00±0,00
DM	2.50±0,00	2,50±0,00
%Asia (-2979,16)	100,06±0,00	100,06±0,00

Cuadro 5.3: Medias de 10 ejecuciones para Alarm y EDAs continuos en el problema del aprendizaje estructural.

	MIMIC_c	UMDA_c
Evaluación	-14463,80±213,23	-14463,13±74,91
DH	10,00±7,40	10,30±4,61
DE	2,90±0,49	2,60±0,44
DM	13,15±12,15	13,25±5,51
%Alarm (-14412,69)	99,65±0,01	99,65±0,00

Cuadro 5.4: Medias de 10 ejecuciones para Alarm y EDAs discretos en el problema del aprendizaje estructural.

	MIMIC_d	UMDA_d
Evaluación	-14423,96±27,41	-14453,54±69,45
DH	4.20±0,76	11,40±5,44
DE	1.40±0,44	2,60±0,44
DM	5.50±1,15	15,40±10,29
%Alarm (-14412,69)	99,92±0,00	99,72±0,00

Cuadro 5.5: Medias de 10 ejecuciones para Alarm y el experimento aleatorio en el problema del aprendizaje estructural.

	Experimento aleatorio
Evaluación	-14522,30±140,66
DH	16,70±9,21
DE	3,40±0,64
DM	22,25±6,81
%Alarm (-14412,69)	99,25±0,01

Cuadro 5.6: Mejores puntuaciones encontradas para la red Alarm en el problema del aprendizaje estructural.

MIMIC_c	-14437,23
UMDA_c	-14453,35
MIMIC_d	-14417,86
UMDA_d	-14435,87
Experimento aleatorio	-14499,73

para la red Asia, y en la Figura 5.6 la misma evolución pero para la red Alarm.

El algoritmo de búsqueda con mejores valores de la métrica es, sin duda, MIMIC_d: el test de Mann-Whitney obtiene un p-valor de $p < 0,001$ entre él y el resto de los métodos. Sin embargo, con MIMIC_c no hay diferencias estadísticamente significativas con respecto a UMDA_c, y UMDA_d lo supera con un p-valor de $p = 0,034$. Finalmente, UMDA_d supera UMDA_c con un p-valor de $p = 0,049$.

Hay que tener en cuenta que la red que se obtiene con el algoritmo K2 y un orden dado no tiene porqué ser exactamente igual a la red original con la que se han generado los datos (Alarm y Asia en este caso) una vez el EDA ha terminado. El motivo es que estamos trabajando con bases de datos forzosamente finitas, aun siendo generadas a partir de dichas redes (en nuestros experimentos, por ejemplo, sólo hemos utilizado 3000 casos), y esto hace que carezcamos de la información que sería necesaria para generar la red original. En la Figura 5.7 podemos ver visualmente la evolucion que sufre la estructura a través de cinco muestras de la ejecución de un EDA (después de la propia red original, la Alarm en este caso). La primera muestra tiene una métrica de $-14{,}536,26$ y una distancia mixta de $24,0$. La última muestra

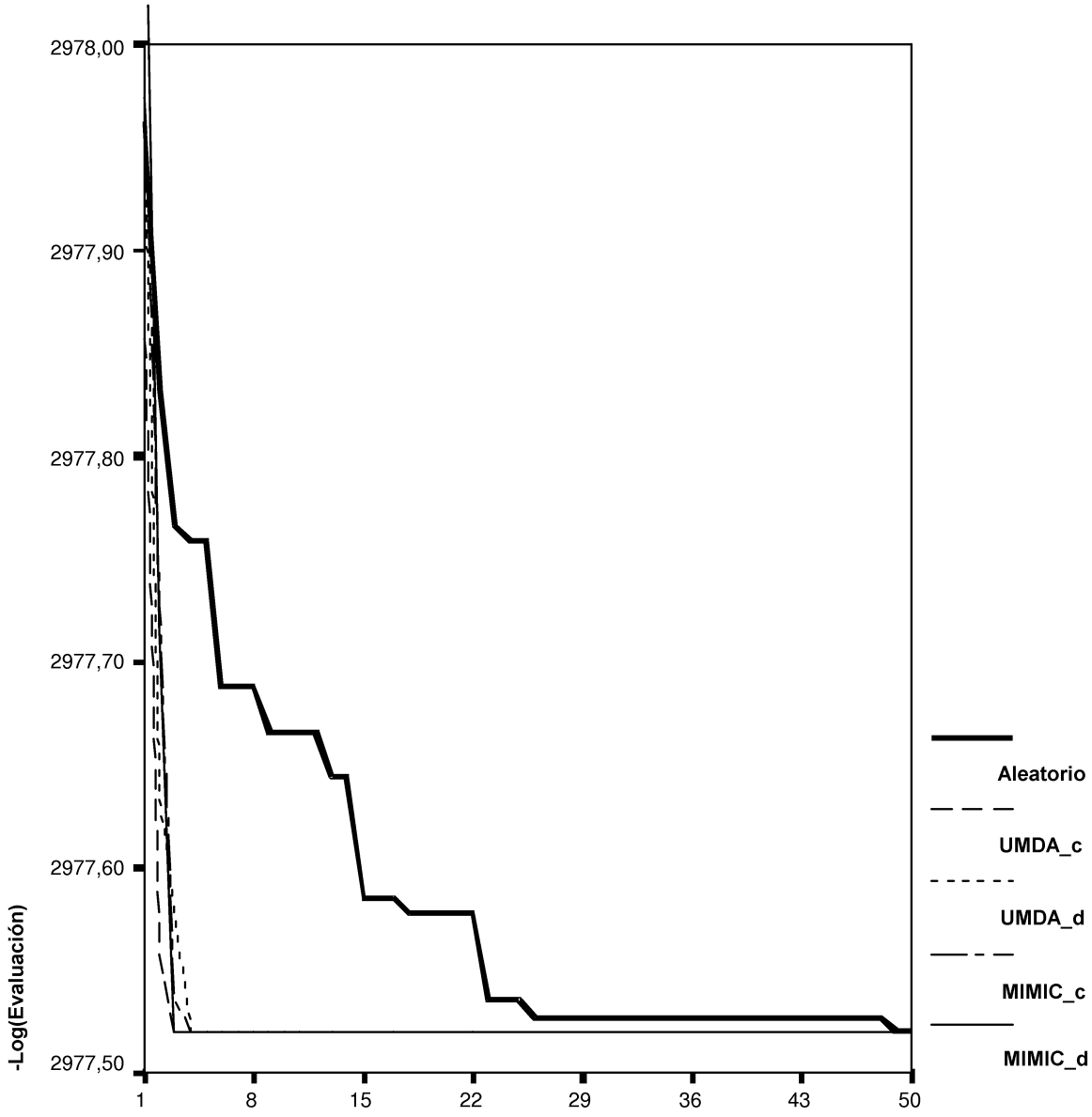

Figura 5.5: Evolución durante las 50 muestras de la evaluación media para la red Asia en el problema del aprendizaje estructural.

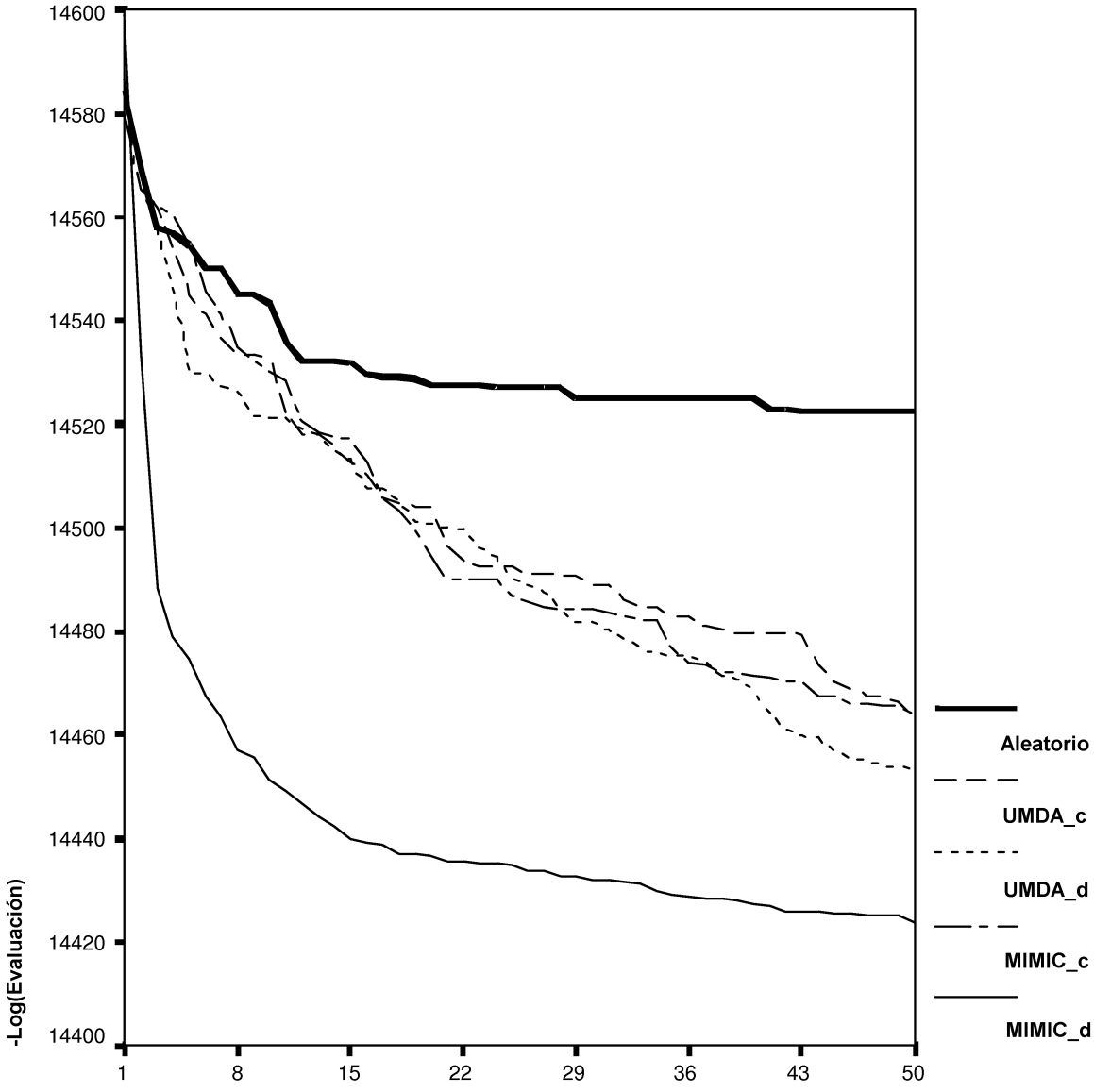

Figura 5.6: Evolución durante las 50 muestras de la evaluación media para la red Alarm en el problema del aprendizaje estructural.

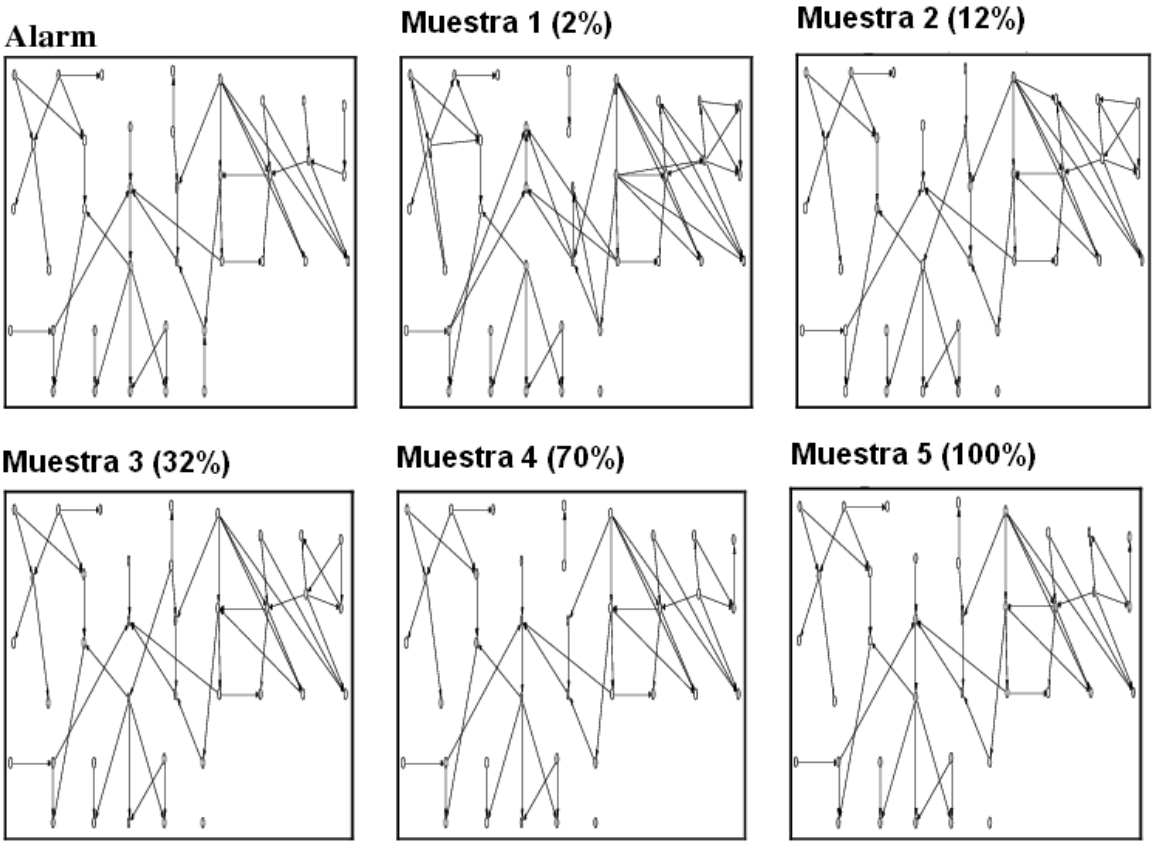

Figura 5.7: Evolución de la estructura de la red devuelta por el K2 a través de cinco muestras para un experimento de Alarm en el problema del aprendizaje estructural. La primera estructura es la propia Alarm. El porcentaje de ejecución del EDA aparece junto a cada muestra.

tiene una métrica de $-14{,}417{,}86$ y una distancia mixta de $4{,}5$.

5.5 Conclusiones

Los experimentos con las redes Alarm y Asia que emplean las estrategias de búsqueda UMDA y MIMIC tanto en dominios continuos como discretos muestran que los EDAs son competitivos con el estado del arte. También queda claro que la representación del individuo es especialmente importante en este caso, sobre todo

si es redundante, esto es, si podemos asociar dos o mas individuos con el mismo orden, que es lo que pasa en el caso continuo, los resultados empeoran claramente. En cuanto al tipo de EDA que ofrece mejores resultados, es sin duda MIMIC en dominios discretos.

Capítulo 6

Inferencia Abductiva Parcial en Redes Bayesianas

'Existe el destino, la fatalidad y el azar; lo imprevisible y, por otro lado, lo que ya está determinado. Entonces como hay azar y como hay destino, filosofemos.'

Séneca

El uso más común que se da a las redes Bayesianas consiste en usarlas para hallar la distribución de probabilidad a posteriori de ciertas variables cuando se ha observado alguna *evidencia*, es decir, el valor de algunas de ellas. Es en este caso cuando se usa la propagación de la evidencia, que tratamos más extensamente en el capítulo 4 dedicado a la triangulación.

Sin embargo, dado un conjunto de valores observados en una red Bayesiana, podemos preguntarnos por las K configuraciones más probables del resto de nodos. Es decir, contamos con una serie de variables cuyo valor hemos observado, y deseamos saber cual es el estado más probable del resto de la red. A este problema se le llama *Inferencia Abductiva Total*, *Explicación Más Probable* o EMP (MPE en inglés) (116). Una variación, o más bien generalización, del problema anterior consiste en hallar el valor no para todas las variables no observadas, sino sólo para parte de ellas. Entonces hablamos de la *Inferencia Abductiva Parcial* o del *Problema del Máximo a Posteriori* (MAP). Una aplicación directa de la solución a este problema sería la determinación de enfermedades a partir de los síntomas. Este problema es NP-duro y por tanto un buen candidato a ser tratado con los paradigmas de la Inteligencia Artificial, dada la imposibilidad de afrontar una solución exacta cuando el tamaño

de la red es grande. Otras aplicaciones del razonamiento abductivo pueden ser la interpretación de imágenes (86), la comprensión del lenguaje natural (146), etc.

Este capítulo, que recoge el trabajo publicado por de Campos, Gámez, Larrrañaga, Moral y Romero en (34), expondrá los resultados de la aplicación de los EDAs al problema del MAP, esto es, trataremos de, dado el valor observado de un conjunto de variables de la red llamado *evidencia*, hallar el valor más probable de otro conjunto de variables, llamado *conjunto de explicación*. Dividiremos el capítulo en las siguientes secciones:

- En la Sección 6.1 se expone brevemente en que consiste el problema, formalizándolo.
- En la Sección 6.2 se exponen otros intentos de solucionarlo.
- En la Sección 6.3 se proponen varios tipos de EDA para resolverlo.
- En la Sección 6.4 se presentan los resultados obtenidos con los EDA y se comparan con los de otros métodos.
- Finalmente, en la Sección 6.5 se exponen las principales conclusiones a las que los resultados obtenidos dan lugar.

6.1 La inferencia abductiva parcial

Sea $\mathbf{X} = (X_1, ..., X_n)$ una variable n-dimensional cuya distribución de probabilidad ha sido factorizada mediante una red Bayesiana. Sea $\mathbf{x}_0$ una instanciación de un conjunto de variables $\mathbf{X}_0$, que llamaremos *evidencia* u *observación*. Sea $\mathbf{X}_N$ el conjunto de variables tal que $\mathbf{X}_N = \mathbf{X} \backslash \mathbf{X}_0$, esto es, el conjunto de las variables que no están instanciadas. Dada dicha red Bayesiana podemos preguntarnos, básicamente, por tres tipos de cuestiones:

- *La Propagación de la Evidencia*. En este uso de la formalización del problema, implícito en la propia red Bayesiana, nos preguntamos por el valor que tienen las variables de $\mathbf{X}_N$, esto es, $P(X_i|\mathbf{X}_0 = \mathbf{x}_o)$ para todo $X_i \in \mathbf{X}_N$.
- *Inferencia Abductiva Total*. También conocido como el problema de la explicación más probable (116). En este caso buscamos el estado de la red que con más probabilidad puede desembocar en la instanciación de variables observadas. Formalmente, buscamos la instanciación $\mathbf{x}_N^*$ de $\mathbf{X}_N$ tal que:

$$\mathbf{x}_N^* = argmax_{\mathbf{x}_N} P(\mathbf{x}_N|\mathbf{X}_0 = \mathbf{x}_o). \tag{6.1}$$

- *Inferencia Abductiva Parcial* o problema del máximo a posteriori (112). Ahora ya no nos preguntamos por una instanciación de $\mathbf{X}_N$, sino por la de un $\mathbf{X}_E \subseteq \mathbf{X}_N$. A $\mathbf{X}_E$ lo llamaremos *Conjunto de Explicación*. Formalmente, buscamos la instanciación que cumpla:

$$\mathbf{x}_E^* = argmax_{\mathbf{x}_E} P(\mathbf{x}_E|\mathbf{X}_0 = \mathbf{x}_o) = argmax_{\mathbf{x}_E} \sum_{\mathbf{x}_R} P(\mathbf{x}_E, \mathbf{x}_R|\mathbf{X}_0 = \mathbf{x}_o) \tag{6.2}$$

  donde $\mathbf{X}_R = \mathbf{X}_N \backslash \mathbf{X}_E$

Hay que tener en cuenta que los tres problemas anteriores son diferentes entre si. Por ejemplo, la inferencia abductiva parcial no se reduce a hacer caso omiso, en la solución hallada para la abducción total, de las variables que no se encuentran en $\mathbf{X}_E$, ya que no tener en cuenta esas variables implica desde el primer momento diferentes pesos del resto de ellas en la solución final.

6.1.1 Lógica de la inferencia abductiva

La abducción puede ser vista como una regla de inferencia. Por ejemplo, la regla de inferencia *Modus Ponens*, una regla básica de la deducción natural (52; 40) sigue el siguiente esquema:

$$\frac{X \rightarrow Y, X}{Y} \tag{6.3}$$

Esto es, está probado que X implica materialmente a Y, y hemos observado que se ha dado el suceso X, por lo que podemos deducir que también se da el suceso Y. El hecho de que se da Y es una consecuencia lógica en el *modus ponens*. Por ejemplo, si hemos probado necesariamente que todos los informáticos tienen barba, entonces si yo soy informático forzosamente tengo que tener barba. Esto en la lógica de predicados se escribe:

$$\frac{\forall X(I(X) \rightarrow B(X)), I(yo)}{B(yo)} \tag{6.4}$$

donde el predicado $I(X)$ significa que X *es informático* y $B(X)$ significa que X *tiene barba*.

En cambio, en la abducción parcial, no hablamos de consecuencias lógicas. El hecho de que se dé $\mathbf{x}_E^*$, que en este caso es la deducción, no se sigue necesariamente de que se dé $\mathbf{x}_0$ y de la existencia de la distribución de probabilidad concreta de $\mathbf{X}$. Por eso, en los dos tipos de inferencia abductiva, nos van a interesar no sólo la instanciación con máxima probabilidad, sino las K instanciaciones máximo probables, ya que dichas instanciaciones no son sino hipótesis entre las cuales habremos de elegir la que más nos convenga. En el caso de la inferencia abductiva, la regla se escribiría:

$$\frac{\forall X(I(X) \rightarrow B(X)), B(yo)}{I(yo)} \tag{6.5}$$

Es decir, tenemos una regla general ($\forall X(I(X) \rightarrow B(X))$, un hecho ($B(yo)$) y una *hipótesis* ($I(yo)$). Como vemos, aunque también tenemos una instanciación o caso particular como conclusión, no se trata de una conclusión definitiva. La regla de inferencia general se puede escribir de la siguiente manera:

$$\frac{\psi \rightarrow \omega, \omega}{\psi} \tag{6.6}$$

Al fijarnos entonces en las K mejores explicaciones, habremos de elegir la que de entre ellas parezca la mejor, es decir, vamos a tener una fase de *selección de hipótesis* posterior a la fase de *generación de hipótesis*, que normalmente se lleva a cabo realizando algún tipo de medida de simplicidad, esto es, normalmente nos van a interesar las hipótesis más simples.

6.1.2 Definición de sistema abductivo

Un sistema abductivo consiste en:

- Una teoría lógica T definida sobre un lenguaje L.
- Un conjunto de sentencias A, escritas también según L llamadas *abducibles*.

T es un conjunto de sentencias válidas escritas en el lenguaje L, esto es, L especifica en qué casos podemos hablar de una sentencia correctamente escrita. Por ejemplo, una sentencia de T podría ser $\forall x SoyFeliz(x) \wedge \neg SoyRico(x) \Rightarrow ExisteDios$. En este caso L es el lenguaje de lógica de predicados.

Dada una sentencia s, tratamos de derivar lógicamente esta sentencia a partir de las sentencias de $T \cup e$, donde e, llamado *explicación* es una sentencia abducible (perteneciente a A) que además es consistente con T, esto es, que añadiéndola no vamos a conseguir derivar una sentencia que sea la negación de alguna que ya pertenezca a T. A es simplemente un método de restricción del tipo de sentencias que podemos utilizar para explicar s. Como hemos visto más arriba, la implicación en T tiene dos interpretaciones distintas según el uso que se hace de ellas: en los predicados abductivos estamos hablando de una especie de plausible relación causa-efecto, mientras que en el resto lo que tenemos es una implicación material.

6.1.3 Generación y selección de posibles explicaciones

Como hemos visto en la sección anterior, tratamos de aumentar las sentencias de T de manera que podamos explicar s. Hay varios métodos para la generación de hipótesis. La mayoría de las veces se opta por la resolución lineal (58; 139; 48). La idea es partir de la transformación de los predicados de T en claúsulas. Después, teniendo en cuenta todas las claúsulas de s (usándolas como claúsulas tope en el árbol de derivación) y aplicando en cada paso de la resolución alguna de las claúsulas de T (o alguna claúsula antecesora en el árbol de derivación), se obtienen otras claúsulas. La resolución lineal acaba cuando llegamos a callejones sin salida, esto es, claúsulas que ya no se pueden resolver con ninguna otra claúsula, ni antecesora ni de T. La reunión de todos los callejones sin salida forma el conjunto de hipótesis. Por ejemplo, en (48), en vez de partir de los elementos de s, se parte de los elementos de $\neg s$. La negación de los callejones sin salida se reúnen al final en una conjunción, que es el conjunto de hipótesis.

Tras la fase de generación de hipótesis, hemos de seleccionar entre ellas las más interesantes dependiendo del uso que queremos hacer de la abducción. Normalmente, el criterio a utilizar será la simplicidad sintáctica, sobre todo si queremos tratar el problema computacionalmente, ya que otros criterios, como por ejemplo la simplicidad psicológica, pueden llegar a ser muy subjetivos. También podemos usar una métrica que favorezca determinadas redes por su topología en detrimento de otras.

6.1.4 Abducción en redes Bayesianas

Trasladando la definición de abducción total que hemos realizado desde el lenguaje de la lógica al de las redes Bayesianas, tenemos que dada la red Bayesiana $G = (\mathbf{X}, L)$ y una observación $\mathbf{x}_0$ del conjunto de variables $\mathbf{X}_0$, entonces $\mathbf{x}_N^* =$

$argmax_{\mathbf{x}_N} P(\mathbf{x}_N|\mathbf{X}_0 = \mathbf{x}_o)$ sería la explicación más probable, es decir, nuestro criterio de selección va a ser el valor de la probabilidad a posteriori de la explicación. En realidad, cualquier instanciación $\mathbf{x}_N$ con una probabilidad a posteriori mayor que cero puede ser considerada una explicación, pero obviamente el conjunto de estas, así consideradas, puede ser extremadamente grande y habremos de seleccionar las mejores.

En el caso de la abducción parcial nos interesa, como hemos visto anteriormente, la instanciación $\mathbf{x}_E^*$ tal que:

$$\mathbf{x}_E^* = argmax_{\mathbf{x}_E} P(\mathbf{x}_E|\mathbf{X}_0 = \mathbf{x}_o) = argmax_{\mathbf{x}_E} \sum_{\mathbf{x}_R} P(\mathbf{x}_E, \mathbf{x}_R|\mathbf{X}_0 = \mathbf{x}_o) \tag{6.7}$$

A $\mathbf{X}_E$ se le llama *conjunto de explicación* o de variables de interés, y a $\mathbf{x}_E^*$ se le llama la *explicación más probable* o *EMP* de $\mathbf{x}_0$. Sea a partir de ahora $P(\mathbf{x}_E|\mathbf{X}_0 = \mathbf{x}_o) = \rho(\mathbf{x}_E|\mathbf{x}_o)$.

La obtención de los K mejores $\mathbf{x}_i^*$ (de los cuales $\mathbf{x}_E^*$ sería la mejor instanciación) se puede realizar por métodos exactos. En principio, una vez generada la distribución de probabilidad conjunta, bastaría con buscar la configuración de máxima probabilidad. Sin embargo, para que estos métodos sean afrontables computacionalmente con redes incluso de tamaño medio, es necesario limitar en algún sentido la estructura de la red. En la abducción parcial, este problema se agudiza debido al sumatorio implícito en su definición.

En el caso de la abducción total, el individuo que nos va a interesar está compuesto por aquellas variables cuya instanciación desconocemos y cuya configuración más probable queremos conocer, y que es por tanto de tamaño $|\mathbf{X}_N|$. Como hemos visto en capítulos anteriores, en los EDA trabajamos con *individuos* que hay que evaluar de alguna manera. Ya podemos adelantar que los individuos, en la abducción total, estarán compuestos por los valores de las variables de $\mathbf{X}_N$, y en la abducción parcial, que es la que nos ocupa en este capítulo, por las variables de $\mathbf{X}_E$. ¿Cómo podríamos evaluar el individuo en el caso de la abducción total? Lo que nos interesa maximizar, como hemos visto antes, es el valor $\rho(\mathbf{x}_N|\mathbf{x}_o)$. Esta expresión es proporcional a $\rho(\mathbf{x}_N, \mathbf{x}_o)$ por lo que podemos usarla para evaluar al individuo $\mathbf{x}_N$. Este valor, para la abducción total, vendría dado por la expresión:

$$\rho(\mathbf{x}_N, \mathbf{x}_o) = \prod_{i=1}^{n} \rho(\mathbf{x}_i|\pi_i) \tag{6.8}$$

que se transforma, para la abducción parcial, en un sumatorio sobre las variables $\mathbf{X}_R = \mathbf{X} \backslash (\mathbf{X}_E \cup \mathbf{X}_0)$. Ahora el tamaño del individuo es menor ($|\mathbf{X}_E|$), pero esto no simplifica los cálculos, ya que la expresión a evaluar es más compleja:

$$\rho(\mathbf{x}_E, \mathbf{x}_o) = \sum_{\mathbf{x}_R} \rho(\mathbf{x}_E, \mathbf{x}_0, \mathbf{x}_R) \tag{6.9}$$

Esto es, tenemos que calcular el sumando de la ecuación 6.9 tantas veces como posibles instanciaciones hay de $\mathbf{X}_R$. Si denotamos este número con $\Omega_{\mathbf{X}_R}$, tenemos que para evaluar una posible explicación $\mathbf{X}_E$ es necesario hacer $\Omega_{\mathbf{X}_R}$ sumas y $|\mathbf{X}| \cdot \Omega_{\mathbf{X}_R}$ multiplicaciones. Si $|\mathbf{X}_E| = 20$ y $|\mathbf{X}_R| = 10$ y cada variable de $\mathbf{X}$ tiene, por ejemplo, 3 valores (compárese esto con la red Alarm, que tiene hasta 37 variables y nodos de hasta 4 valores posibles) tenemos $20 \cdot 3^{10} = 1180980$ multiplicaciones y $3^{10} = 59049$ sumas. Con redes en las que el número de valores por nodo sea mayor y que tengan un conjunto de explicación reducido, el cálculo de la evaluación para las $\Omega_{\mathbf{X}_R}$ posibles instanciaciones es inabordable.

6.2 Métodos de búsqueda de hipótesis más probables

A continuación veremos algunos métodos, tanto exactos como heurísticos, para realizar este cálculo lo más eficientemente posible.

6.2.1 Métodos exactos para la abducción

En el caso de los métodos exactos, la necesidad de obtener la solución en un tiempo razonable obliga a diversas restricciones sobre la red Bayesiana o sobre el problema en sí.

Por ejemplo, Cooper desarrolla un algoritmo de búsqueda eficiente en (44) llamado *NESTOR* que es capaz de obtener las mejores explicaciones sin llegar a examinar todas las posibles. Para ello no visita aquellas explicaciones que van a ser peores que la mejor en curso, posibilidad que detecta a costa de hacer que todas las variables del conjunto de abducibles $\mathbf{X}_E$ tengan sólo dos valores. Además, si $\mathbf{X}_I$ y $\mathbf{X}_J$ son dos conjuntos de variables abducibles y $\mathbf{x}'_I$ y $\mathbf{x}'_J$ sus instanciaciones escogiendo el primer valor para todas las variables abducibles, se tiene que cumplir que $P(\mathbf{x}'_I \bigcup \mathbf{x}'_J) \leq P(\mathbf{x}'_I)$ y $P(\mathbf{x}'_I \bigcup \mathbf{x}'_J) \leq P(\mathbf{x}'_J)$. Como vemos, hablamos aquí de un contexto muy restringido. Peng mejora la eficiencia en (121) con un algoritmo similar, pero para ello divide el

conjunto de variables en dos y sólo permite arcos del primero conjunto al segundo, siendo el segundo el de la evidencia.

Li y D´Ambrosio proponen en (99) un algoritmo de orden polinomial sobre n para calcular la próxima instanciación más probable. Primero, utilizan distribuciones parcialmente instanciadas con respecto a una factorización óptima, combinándolas para obtener la primera explicación, y luego aplican un algoritmo de orden lineal para conseguir las $K-1$ posibles explicaciones restantes modificando de manera incremental árboles de evaluación de términos probabilísticos. El problema consiste en la factorización óptima de la red, que es un problema NP-duro y que para redes que no sean simplemente conectadas (esto es, que pueden tener más de un camino posible entre dos nodos) tiene que ser hallada mediante heurísticos. Srinivas y Nayak (145) proponen un algoritmo de complejidad similar que enumera eficientemente las instanciaciones de una red Bayesiana en orden de probabilidad decreciente, pero que tiene el mismo problema cuando trata con redes que no son simplemente conectadas.

Shimony y Charniak (142; 41; 141) proponen transformar la red Bayesiana en un grafo pesado acíclico AND/OR (GPAAO) asociado a un sistema de reglas basado en costes, en los que los nodos representan proposiciones y los arcos relaciones lógicas. En este tipo de grafos, dado un conjunto de hechos, cabe preguntarse por el coste mínimo asociado a las reglas que se usan para probar dicho conjunto. Los autores demuestran que hay una correspondencia entre hallar dicho coste mínimo y el MAP para la red Bayesiana asociada (en el que la evidencia es exactamente el conjunto de hechos a probar en el GPAAO). Sin embargo, la complejidad espacial del GPAAO crece exponencialmente con el tamaño de la red Bayesiana asociada. Santos y col. (133) proponen una mejora transformando el problema de la minimización del coste en una solución óptima para un programa lineal de enteros $0-1$, que puede ser alcanzada en un tiempo polinomial si el GPAAO cumple una propiedad llamada unimodularidad total.

Los algoritmos de propagación de la evidencia para la obtención de las probabilidades marginales son muy eficientes, por lo que se han propuesto también algoritmos basados en ellos para el problema del MAP. La idea de los métodos de propagación consiste en aprovechar las (in)dependencias de la red para que cuando el valor de un nodo cambia, sólo se vean implicados en un nuevo cálculo sus nodos vecinos y así no haya que tener en cuenta todos los nodos de la red en el mismo. En el capítulo 4 se habla de varios algoritmos de propagación concretos. Para aplicar la propagación al problema del MAP, se sustituye en la marginalización la suma por el máximo (gracias a la propiedad distributiva del máximo con respecto a la multiplicación).

En el algoritmo propuesto en (51), Dawid se basa en el algoritmo de propaga-

ción Hugin, basado a su vez en los trabajos de Jensen y col. (81; 82). Como hemos comentado antes, la marginalización de un conglomerado a su separador se calcula:

$$\psi(S_i) \leftarrow \psi(C_i)^{\downarrow S_i} = max_{C_i \setminus S_i} \psi(C_i) \tag{6.10}$$

Si X_E forma un subárbol del árbol de conglomerados, entonces el algoritmo para la abducción total puede ser aplicado a la abducción parcial, realizando primero una propagación por suma hacia el subárbol y aplicando luego al subárbol cualquier método para la abducción total. Nilsson mejora en (113) este algoritmo limitando la propagación a la fase ascendente.

En (136), Seroussi y Goldmard utilizan el árbol de juntura (que se obtiene tras la moralización, triangulación y el ordenamiento de los conglomerados). El principio de su algoritmo consiste en visitar de abajo a arriba cada conglomerado del árbol y almacenar en cada nivel las K instanciaciones más probables de los niveles más profundos. Sin embargo, la complejidad del algoritmo depende del máximo tamaño de conglomerado de la red y a pesar de que es capaz de devolver las K configuraciones más probables para cualquier K, su coste computacional puede ser elevado.

6.2.2 Métodos heurísticos

De Campos y col. (35) proponen un algoritmo genético para solucionar el problema. Proponen una modificación del *modGA* de Michalewicz (106), conservante, generacional y con selección elitista. Se mantienen el 50 % de los mejores cromosomas de una generación a la siguiente para mantener la diversidad y evitar convergencias prematuras, y para el otro 50 % se generan nuevos individuos, un 35 % con el operador de cruce de dos puntos y un 15 % mediante mutación con mayor probabilidad sobre los mejores individuos, y siempre sobre el mejor (los porcentajes están seleccionados mediante experimentación). Como individuo, consideran el cromososa $\mathbf{x}_E$, y lo evalúan usando $\rho(\mathbf{x}_E, \mathbf{x}_0)$, donde $\mathbf{x}_o$ es la observación. Esta estrategia será también la que utilicemos en este capítulo para evaluar la bondad de un individuo propuesto por el EDA. La evaluación del cromosoma se lleva a cabo considerando su árbol de conglomerados o de juntura $T = C_1, ..., C_t$ con una única raiz (C_1). Se comienza insertando la evidencia $\mathbf{x}_0$ en T y luego la instanciación propuesta por el genético $\mathbf{x}_E$, y se realiza después una propagación de la evidencia desde las hojas hasta la raiz del árbol, devolviéndose como valor de $\rho(\mathbf{x}_E, \mathbf{x}_0)$ la suma de los potenciales almacenados en C_1. Además, los autores proponen un podado del árbol de conglomerados que acelera aún más el cálculo evitando la repetición de determinados cálculos.

Gámez propone en (69) la aplicación del enfriamiento estocástico con buenos resultados. Este heurístico tiene un comportamiento similar al de los métodos de ascensión de colinas, pero para evitar quedar atascado en un mínimo/máximo local, prueba a seguir por un camino que sabe a ciencia cierta que en principio es peor. Esta decisión se hace en base a un parámetro llamado *temperatura* que al principio tiene un valor alto (luego hay muchas posibilidades de explorar caminos a priori malos) y va decrementándose con el tiempo mediante lo que se llama un *programa de enfriamiento*. El uso de la temperatura se hace mediante el criterio de Boltzmann, es decir, se opta por un camino peor con una probabilidad de $e^{(-C(\mathbf{x}')-C(\mathbf{x}))/t}$, donde t es la temperatura, $\mathbf{x}$ es una configuración del espacio de búsqueda y $\mathbf{x}'$ es la configuración peor. $C(\mathbf{x})$ es la función de coste que nos dice cuán mejor/peor es una determinada configuración, y es equivalente a la utilizada en (35).

6.3 Solución del problema con EDAs

La solución del problema con EDAs es muy similar a la propuesta para el caso de los algoritmos genéticos por de Campos y col. (35). Simplemente, se sustituyen los operadores de cruce y mutación por la simulación de la estimación de la distribución de la probabilidad en cada generación, siendo la representación del individuo idéntica, esto es, el *cromosoma* $\mathbf{x}_E$ pasa a ser el *individuo* $\mathbf{x}_E$. La función de evaluación es exactamente la misma. Formalicemos ahora un tanto el problema.

Definición 6.3.1 *Sean la red Bayesiana $G = (\mathbf{X}, L)$ y una instanciación observada $\mathbf{x}_0$ del conjunto de variables $\mathbf{X}_0 \in \mathbf{X}$. Llamaremos a $\mathbf{x}_N^* = argmax_{\mathbf{x}_N} P(\mathbf{x}_N|\mathbf{x}_o)$ la explicación más probable en G dada la evidencia $\mathbf{x}_o$.*

Como vemos, para evaluar un individuo $\mathbf{x}_N$, hemos de calcular $P(\mathbf{x}_N|\mathbf{x}_o)$ y tratar de maximizar ese valor para obtener el mejor.

Teorema 6.3.1 *Dado que $P(\mathbf{x}_N|\mathbf{x}_o) \propto P(\mathbf{x}_N, \mathbf{x}_o)$, maximizar $P(\mathbf{x}_N|\mathbf{x}_o)$ es equivalente a maximizar:*

$$P(\mathbf{x}) = P(\mathbf{x}_N, \mathbf{x}_o) = \prod_{i=1}^{n} P(\mathbf{x}_i|\mathbf{\Pi}_i) \tag{6.11}$$

Definición 6.3.2 *Llamaremos a $\mathbf{x}_E^*$ la explicación más probable en G del conjunto de explicación $\mathbf{X}_E$ dada la evidencia $\mathbf{x}_o$ y la definiremos como:*

$$\mathbf{x}_E^* = argmax_{\mathbf{x}_E} P(\mathbf{x}_E|\mathbf{x}_o) = argmax_{\mathbf{x}_E} \sum_{\mathbf{x}_R} P(\mathbf{x}_E|\mathbf{x}_R|\mathbf{x}_o) \tag{6.12}$$

Teorema 6.3.2 . *Tal y como hemos procedido con la abducción total, dado que también se da que* $P(\mathbf{x}_E, \mathbf{x}_R|\mathbf{x}_o) \propto P(\mathbf{x}_E, \mathbf{x}_R, \mathbf{x}_o)$*, maximizar* $\sum_{\mathbf{x}_R} P(\mathbf{x}_E, \mathbf{x}_R|\mathbf{x}_o)$ *es equivalente a maximizar* $\sum_{\mathbf{x}_R} P(\mathbf{x}_E, \mathbf{x}_R, \mathbf{x}_o)$

Ya hemos comentado antes que la expresión que aparece en 6.3.2 puede llegar a ser muy costosa computacionalmente en redes grandes, por lo que aprovecharemos la factorización implícita en la red para calcular la función de evaluación $P(\mathbf{x}_E, \mathbf{x}_o)$ a partir del árbol de conglomerados. El algoritmo que calcula la evaluación se comporta de la siguiente manera:

- Introduce la evidencia $\mathbf{x}_o$ en el árbol de conglomerados.
- Introduce, como si fuera también una evidencia, el individuo $\mathbf{x}_E$ propuesto por el EDA y del que queremos hallar su evaluación.
- Realiza una propagación de la evidencia similar a la absorción de la evidencia para el algoritmo de Lauritzen y Spielgelhalter del que hablábamos en el capítulo 4. En este caso, se utiliza el algoritmo Hugin (también comentado en el mismo capítulo), pero limitado a la fase de ascenso de las hojas a la raíz del árbol. $P(\mathbf{x}_E, \mathbf{x}_o)$ es entonces la suma de los potenciales almacenados en el conglomerado raíz C_1.

En (35) se muestra cómo el cálculo de la propagación puede ser más eficiente podando el árbol para un conjunto de explicación determinado, lo que hace que no se realicen cálculos innecesarios cuando se evalúa un nuevo individuo.

6.3.1 Parametrización del EDA

Para este problema hemos empleado los algoritmos UMDA (107), MIMIC (23) y EBNA (59), cuyas descripciones pueden encontrarse en el Capítulo 2, y que representan cada uno un incremento con respecto al anterior de la complejidad de la factorización de la distribución de probabilidad de los individuos del grupo de selección. Sólo utilizaremos individuos discretos, que estarán compuestos, como en el trabajo de de Campos y col. (35) por el subconjunto de las variables no instanciadas que nos interesan (X_E). La generación inicial se obtiene aleatoriamente. Cada lanzamiento del EDA está parametrizado de la siguiente manera:

- Criterio de parada - El criterio de parada no está relacionado con ningún criterio de convergencia. Normalmente, el criterio de convergencia, sea cual sea el

método, no asegura la terminación del algoritmo. Los experimentos imprimen una muestra (los $K = 50$ individuos mejores encontrados hasta entonces y su evaluación) cada aproximadamente 500 individuos distintos evaluados, y se ha decidido que el programa termine una vez impresas 10 muestras, esto es, una vez evaluados más de 5.000 individuos. La cifra no puede ser exacta debido a que es necesario esperar a que acabe la generación en curso.

- Grupo de seleccion - Siempre se seleccionan los $N = 50$ mejores individuos encontrados hasta entonces.

- Nueva población - Una vez seleccionados los N mejores individuos y creada la red a partir de ellos, se simulan tantos individuos nuevos como el tamaño de la población (M). En cada simulación de un nuevo individuo se rechazan aquellos que ya hayan aparecido en alguna generación previa desde el comienzo de la evolución. Si hay 50 intentos no válidos seguidos de simular un individuo nuevo, el individuo repetido se añade a la población y se sigue simulando el resto. Una vez obtenidos M individuos nuevos, se seleccionan los M mejores de entre los M nuevos y los N mejores de la generación anterior que se utilizaron para generar la red Bayesiana.

- Otros parámetros - No hay elitismo y la selección de los individuos se realiza por truncación.

6.3.2 Redes utilizadas

Hemos utilizado una red bien conocida, Alarm (16), descrita en el Capítulo 2 y dos redes generadas artificialmente: *random100* y *random100e*. Mientras que Alarm tiene 37 nodos, las redes artificiales llegan hasta 100. Para las dos últimas se ha seguido el mismo procedimiento a la hora de generar su estructura, pero mientras que para la primera se han utilizado números aletorios uniformes para generar las tablas de probabilidad, para la segunda se ha seguido un procedimiento más complejo: a partir de dos números aleatorios uniformes x e y, la probabilidad de los dos valores (de las marginales para los nodos raiz y las condicionales para el resto) de una variable se determinan normalizando con x^5 e y^5, lo cual da lugar a probabilidades extremas. En el Cuadro 6.1 aparecen las características de estas redes.

Cuadro 6.1: Características de las redes utilizadas para el problema de la abducción parcial. Mínimo, máximo y media hacen referencia a, respectivamente, el tamaño mínimo de la tabla de probabilidades, la máxima y la media asociada a cada nodo.

Red	Nodos	Arcos	Estados	Mínimo	Máximo	Medio
Alarm	37	46	{2, 3, 4}	2	108	20.3
random100	100	122	2	2	32	5.88
random100e	100	128	2	2	64	6.54

6.3.3 Función de evaluación del individuo

Como función de evaluación utilizaremos, como ya hemos comentado anteriormente, la expuesta por de Campos y col. en (35), esto es, tras insertar la evidencia $\mathbf{x}_0$ en el árbol de conglomerados T y luego la instanciación propuesta por el genético ó el EDA $\mathbf{x}_E$, se realiza después una propagación de la evidencia desde las hojas hasta la raiz del árbol, devolviéndose como valor de $P(\mathbf{x}_E, \mathbf{x}_o)$ la suma de los potenciales almacenados en C_1.

6.3.4 Planteamiento de los experimentos

Hemos lanzado los experimentos para una población de $M = 50, 100, 150, 200, 250, 300, 400$ y 500 individuos para las tres redes, pero mostraremos en este trabajo únicamente los resultados de la población que haya dado mejor resultado para un algoritmo concreto. Por población y red se han realizado 50 experimentos, esto es, 50 lanzamientos de cada EDA para cada tamaño de población y cada red, de manera que se mostrarán las medias de estas 50 ejecuciones. Las pruebas se han llevado a cabo en PCs con procesadores Pentium III y IV con distintas cantidades de memoria RAM por debajo de 1Gb, en los que la ejecución de todas las pruebas llevaba aproximadamente una semana.

Para la determinación de $\mathbf{X}_E$, esto es, del conjunto de explicación, se ha utilizado un método pseudo aleatorio. Primero se fija $|\mathbf{X}_E|$ y luego se generan aleatoriamente diversos conjuntos de $|\mathbf{X}_E|$ variables, eligiéndose siempre aquel que es más complicado de solucionar mediante métodos exactos, exceptuando el caso de los experimentos con las redes *random100*, con las que la solución exacta es inabordable usando el equipo antes mencionado. En el Cuadro 6.2 aparece el tamaño de $\mathbf{X}_E$ empleado para cada red. Con Alarm se han realizado pruebas con varios tamaños de $\mathbf{X}_E$ para comprobar

Cuadro 6.2: Tamaño de $\mathbf{X}_E$ para cada una de las redes usadas en los experimentos junto con el tamaño del conjunto de posibles instanciaciones de $\mathbf{X}_E$ para el problema de la abducción parcial.

Red	$\lvert\mathbf{X}_E\rvert$	$\omega_{\mathbf{X}_E}$
Alarm	18	143.327.232
Alarm	19	214.990.848
Alarm	20	382.205.952
random100	30	1.073.741.824
random100e	30	1.073.741.824

la influencia de este parámetro en el comportamiento de los algoritmos.

Junto con los K mejores individuos en cada muestra, se muestra también su masa de probabilidad. Con $masa'_x$ denotaremos la masa de probabilidad de los x mejores individuos propuestos por el algoritmo. En el caso de Alarm, dado que es posible hallar resultados exactos, se mostrará el porcentaje de masa obtenido ($\%masa'_x$) con respecto a la masa de la solución exacta ($masa_x$):

$$\%masa'_{\mathbf{x}} = \frac{masa'_{\mathbf{x}} * 100}{masa_{\mathbf{x}}} \tag{6.13}$$

6.4 Resultados obtenidos

En las Figuras 6.1, 6.2 y 6.3 podemos ver la evolución a lo largo de la ejecución de los EDA de $\%masa'_1$ para la red Alarm y, respectivamente, el tamaño 18, 19 y 20 de $|\mathbf{X}_E|$. Como elemento de comparación, se muestra también el resultado la ejecución del algoritmo genético citado en el trabajo de de Campos y col. (35). En las Figuras 6.4 y 6.5 se puede ver la evolución, ya de $masa'_x$ directamente, para las redes *random100* y *random100e* respectivamente, con un tamaño de $|\mathbf{X}_E|$ de 30.

En los Cuadros 6.3, 6.4 y 6.5 aparecen los resultados medios junto con la desviación estándar para los experimentos con la red Alarm y los distintos tamaños del grupo de explicación. Como elemento de comparación, se muestra también el resultado para el algoritmo genético antes mencionado.

Los Cuadros 6.6 y 6.7 son los equivalentes para los experimentos con, respectivamente, las redes *random100* y *random100e* y un tamaño de $\mathbf{X}_E$ de 30.

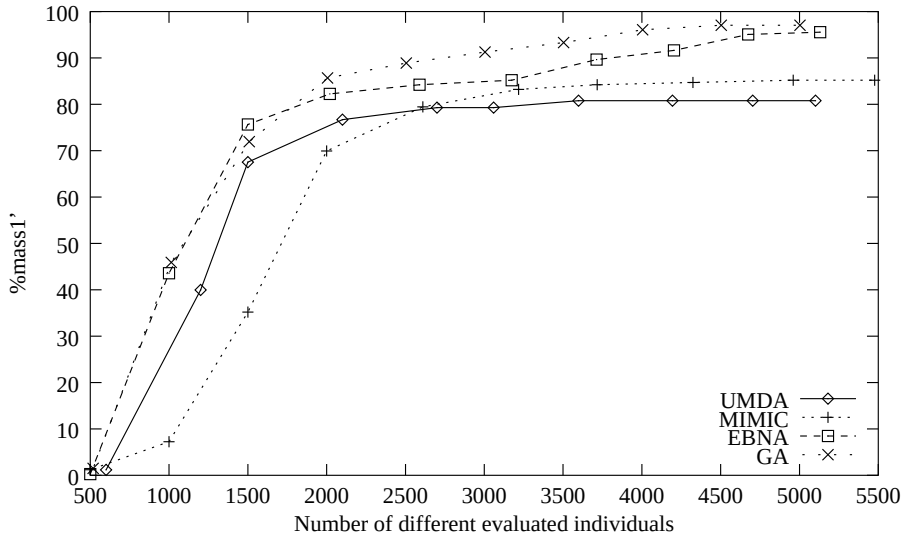

Figura 6.1: Evolución de $\%masa'_1$ para la red Alarm y tamaño 18 de $|\mathbf{X}_E|$.

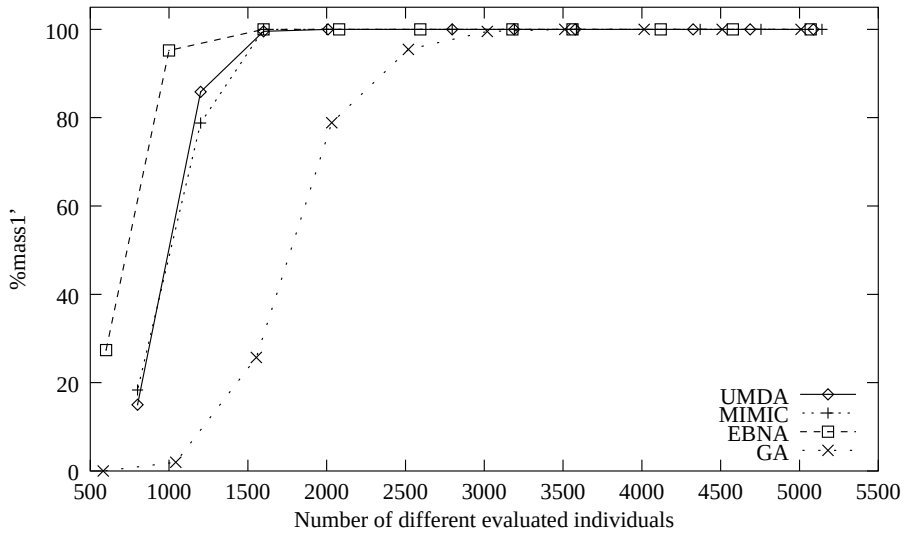

Figura 6.2: Evolución de $\%masa'_1$ para la red Alarm y tamaño 19 de $|\mathbf{X}_E|$.

Cuadro 6.3: Medias de 50 ejecuciones para Alarm y $|\mathbf{X}_E| = 18$ en el problema de la abducción parcial. Los tamaños de las poblaciones escogidas han sido 300 para UMDA, 500 para MIMIC, 250 para EBNA y 100 para el AG.

	$\%masa'_1$	$\%masa_10'$	$\%masa_25'$	$\%masa_50'$
UMDA	80,77±10,32	75,16±12,25	71,07±12,52	68,25±12,43
MIMIC	85,21±12,20	80,43±15,22	76,66±17,07	74,19±17,91
EBNA	95,56±9,57	93,62±12,74	92,44±14,95	91,75±16,21
AG	97,04±0,81	89,63±1,38	85,16±1,88	83,19±2,09

Cuadro 6.4: Medias de 50 ejecuciones para Alarm y $|\mathbf{X}_E| = 19$ en el problema de la abducción parcial. Los tamaños de las poblaciones escogidas han sido 400 para UMDA, 400 para MIMIC, 200 para EBNA y 200 para el AG.

	$\%masa'_1$	$\%masa_10'$	$\%masa_25'$	$\%masa_50'$
UMDA	100,00±0,00	100,00±0,00	100,00±0,00	98,95±0,25
MIMIC	100,00±0,00	100,00±0,00	100,00±0,00	98,99±0,28
EBNA	100,00±0,00	100,00±0,00	100,00±0,00	99,62±0,46
AG	100,00±0,00	100,00±0,00	100,00±0,00	99,38±0,05

Cuadro 6.5: Medias de 50 ejecuciones para Alarm y $|\mathbf{X}_E| = 20$ en el problema de la abducción parcial. Los tamaños de las poblaciones escogidas han sido 500 para UMDA, 500 para MIMIC, 500 para EBNA y 300 para el AG.

	$\%masa'_1$	$\%masa_10'$	$\%masa_25'$	$\%masa_50'$
UMDA	88,07±29,04	81,32±34,75	77,62±35,81	74,54±36,35
MIMIC	87,14±30,28	81,06±34,01	76,67±35,49	73,26±36,02
EBNA	97,76±12,62	96,79±13,12	95,19±14,86	93,80±16,98
AG	99,37±0,21	97,36±0,47	93,00±1,50	87,55±5,33

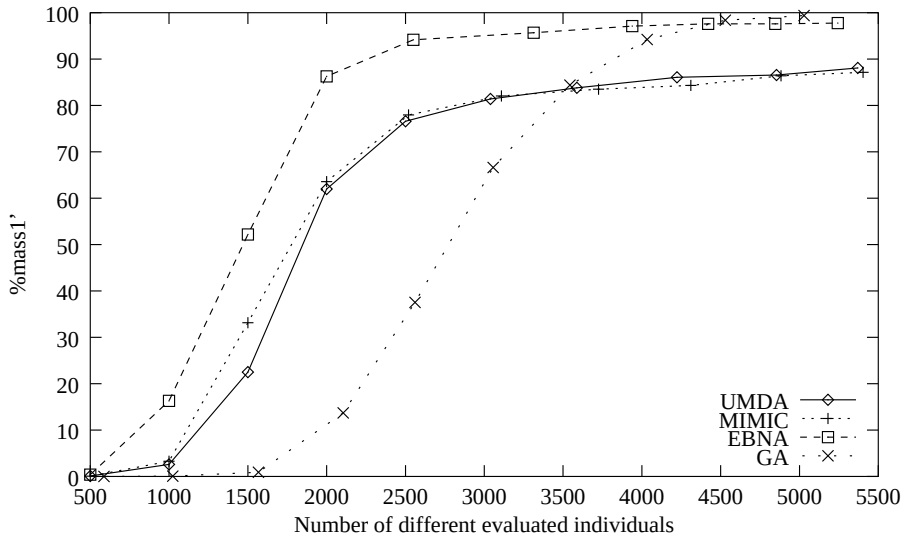

Figura 6.3: Evolución de $\%masa'_1$ para la red Alarm y tamaño 20 de $|\mathbf{X}_E|$.

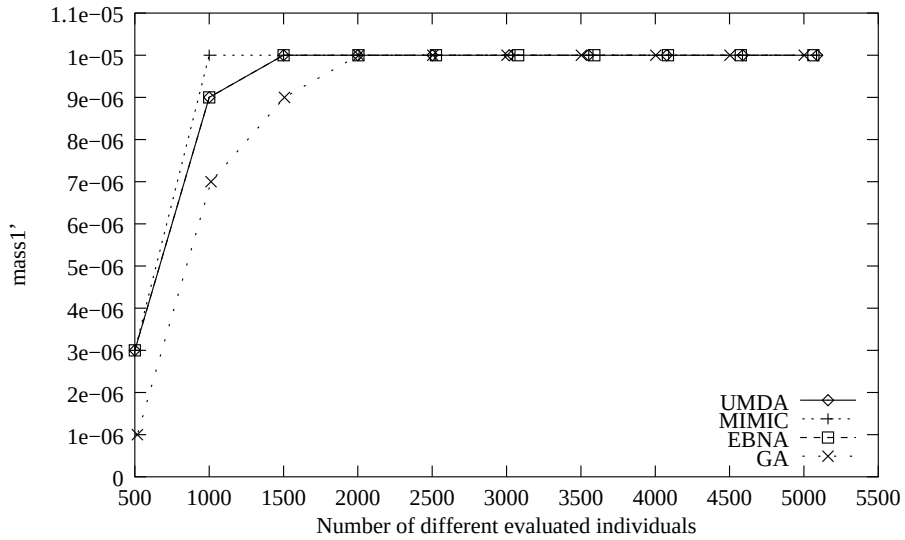

Figura 6.4: Evolución de $\%masa'_1$ para la red *random100* y tamaño 30 de $|\mathbf{X}_E|$.

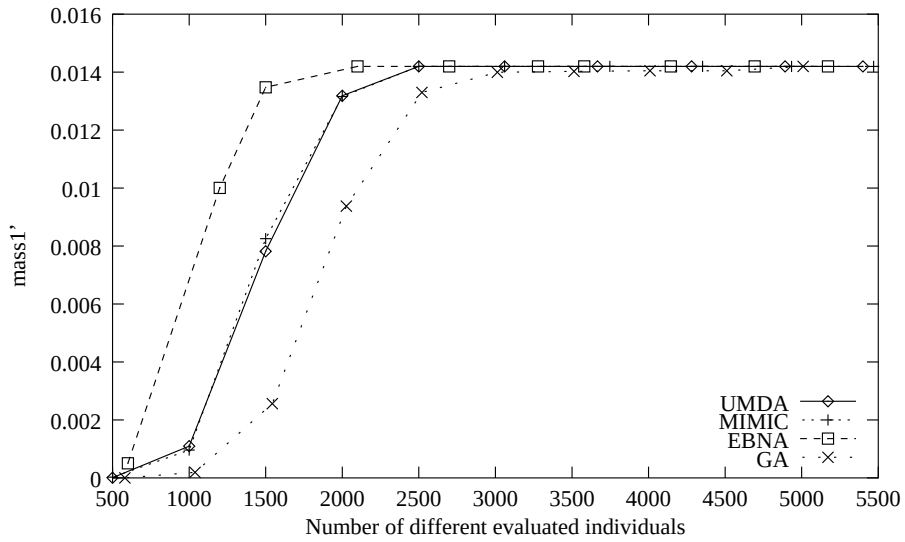

Figura 6.5: Evolución de $\%masa'_1$ para la red *random100e* y tamaño 30 de $|\mathbf{X}_E|$.

Cuadro 6.6: Medias de 50 ejecuciones para random100 y $|\mathbf{X}_E| = 30$ en el problema de la abducción parcial. Los tamaños de las poblaciones escogidas han sido 100 para UMDA, 100 para MIMIC, 100 para EBNA y 100 para el AG.

	$\%masa'_1$	$\%masa_10'$	$\%masa_25'$	$\%masa_50'$
UMDA	0,000010±0,00	0,000077±0,00	0,000167±0,00	0,000295±0,00
MIMIC	0,000010±0,00	0,000077±0,00	0,000167±0,00	0,000295±0,00
EBNA	0,000010±0,00	0,000077±0,00	0,000167±0,00	0,000295±0,00
AG	0,000010±0,00	0,000076±0,00	0,000166±0,00	0,000297±0,00

Los resultados obtenidos por los tres algoritmos EDA y el genético son similares para las redes *random* y para la red Alarm con un tamaño del grupo de explicación de 19. En cambio, para la red Alarm y los valores de $|\mathbf{X}_E|$ 18 y 20, las diferencias son más acusadas. Probablemente esto sea debido a que en los dos últimos casos el espacio de búsqueda sea más complejo que en los primeros. Del análisis de los cuadros podemos extraer las siguientes conclusiones:

- EBNA siempre logra mejores resultados que UMDA y MIMIC. Esto es debido

Cuadro 6.7: Medias de 50 ejecuciones para random100e y $|\mathbf{X}_E| = 30$ en el problema de la abducción parcial. Los tamaños de las poblaciones escogidas han sido 500 para UMDA, 500 para MIMIC, 300 para EBNA y 200 para el AG.

	$\%masa'_1$	$\%masa_10'$	$\%masa_25'$	$\%masa_50'$
UMDA	0,014197±0,00	0,090864±0,00	0,164966±0,002	0,237114±0,006
MIMIC	0,014197±0,00	0,090848±0,00	0,163928±0,002	0,232105±0,007
EBNA	0,014197±0,00	0,091064±0,00	0,168794±0,003	0,247711±0,008
AG	0,014197±0,00	0,091073±0,00	0,168202±0,003	0,244589±0,008

sin duda a que EBNA trata distribuciones de probabilidad sin restricciones, mientras que UMDA y MIMIC están basados en aproximaciones utilizando estadísticas de orden uno y dos respectivamente.

- Los resultados del algoritmo genético son siempre superiores a UMDA y MIMIC en todos los casos.
- Mientras que el genético supera a EBNA cuando tenemos en cuenta únicamente la mejor hipótesis, según K va incrementándose es EBNA el que obtiene los mejores resultados.
- La desviación estándar en cada algoritmo va incrementándose según la secuencia AG, EBNA, UMDA, MIMIC, estando la de los algoritmos EDA siempre bastante por encima de la del genético.
- El comportamiento de los EDA parece estar menos sujeto al tamaño de la población que el del algoritmo genético.
- De las Figuras 6.1, 6.2, 6.3, 6.4 y 6.5 podemos extraer la conclusión de que el número de evaluaciones que requiere un EDA para obtener buenos resultados es menor que el que requiere el algoritmo genético, ya que presenta una convergencia más rápida.

6.5 Conclusiones

El resultado de EBNA es similar al de los algoritmos genéticos para el problema de la abducción parcial, siendo uno u otro más adecuado según estemos hablando de

obtener la mejor hipótesis o de obtener las K mejores. Ambos superan claramente a los otros dos algoritmos EDA (UMDA y MIMIC). En el caso de EBNA, la robustez ante los distintos tamaños de población y el número inferior de evaluaciones para alcanzar resultados aceptables también resulta ser una ventaja con respecto a los genéticos. Sin embargo, la complejidad del algoritmo es mucho mayor y los tiempos de ejecución se incrementan bastante.

Capítulo 7

EDAs Recursivos

'Nunca puedes planear el futuro a través del pasado.'

Edmund Burke

Uno de los problemas que lleva asociado el uso de los algoritmos de estimación de distribuciones es la lentitud de ejecución si se comparan, a igual complejidad del problema, con otras estrategias evolutivas como, por ejemplo, los algoritmos genéticos.

Este capítulo, que recoge el trabajo de Romero y Larrañaga en fase de revisión en (127), expondrá los resultados de la aplicación de la recursividad a los EDAs en un problema en el que estos algoritmos no alcanzan un resultado del todo satisfactorio y que ya se expuso en el capítulo 4: la búsqueda de la triangulación óptima en redes Bayesianas. Dividiremos el capítulo en las siguientes secciones:

- En la sección 7.1 se exponen otros intentos de solucionar el problema de la ineficiencia mediante otros paradigmas de computación.
- En la sección 7.2 se describe brevemente en qué consiste la recursividad.
- En la sección 7.3 se propone un algoritmo EDA recursivo concreto basado en la división sistemática del invidividuo y se hace una primera comparación de su comportamiento con los algoritmos del capítulo 4.
- En la sección 7.4 se presentan los resultados obtenidos con la estrategia recursiva para los mismos tipos de EDA que en el capítulo 4.

- Finalmente, en la sección 7.5 se exponen las principales conclusiones que se pueden extraer de los experimentos realizados, así como una propuesta para paralelizar la ejecución.

7.1 Otros intentos de mejorar la eficiencia

La eficiencia de un determinado algoritmo no suele recibir la atención que merece, cuando en realidad se trata del factor más crítico a la hora de determinar su uso en problemas reales. La eficiencia no sólo nos interesa para obtener la respuesta a un problema dado lo antes posible, sino tambien porque en determinados contextos un algoritmo ineficiente podría ser incapaz de devolver un resultado. Es decir, la inversión en la aceleración de un algoritmo posibilita su aplicación en un conjunto de contextos [1] distintos mayor.

Hay dos maneras de afrontar la mejora de la eficiencia [2] de un algoritmo. Una de ellas es identificar primero sus cuellos de botella y tratar de minimizar su efecto sobre el tiempo de ejecución total aplicando para ello una estrategia de simplificación del contexto que comentaremos brevemente en el siguiente párrafo. Esta es la opción utilizada en los trabajos que analizaremos más adelante. En el caso de los EDA, las fases más costosas computacionalmente hablando son las etapas de aprendizaje de la distribución de probabilidad, la simulación de la red Bayesiana para generar los nuevos individuos (la ejecución del algoritmo PLS en nuestro caso) y normalmente también la evaluación de los individuos si nuestra función objetivo es lo suficientemente compleja. En (17) se puede ver un análisis de la contribución al tiempo de ejecución de las distintas etapas de varios algoritmos EDA, donde se pone de manifiesto que a medida que el aprendizaje es menos restrictivo a la hora de seleccionar los padres de cada nodo (EBNA, EGNA), menos significativo es el porcentaje de ejecución que gastamos en la evaluación de los individuos, siendo el de la etapa de simulación bastante bajo incluso en los EDAs mas restrictivos, como pueden ser UMDA y MIMIC. Es dedir, la etapa de aprendizaje es el verdadero cuello de botella en los EDA.

La otra opción consiste en intentar dividir el problema global P, que se desarrolla en un contexto C, en otros problemas $p_1, p_2, \ldots, p_i$, donde i no tiene porqué estar prefijado a priori, y en los que cada contexto c_i es, de alguna manera, menos complejo

[1] Cuando hablamos de *contexto* nos referimos a las características de los tipos de dato que el algoritmo maneja como entrada, como pueden ser su diseño, su magnitud, etc.

[2] Cuando hablamos de *eficiencia* nos referimos aquí al tiempo de ejecución del algoritmo, y no al espacio en memoria interna o externa utilizado.

que C, y ese decremento de complejidad tiene una influencia positiva a la hora de la ejecución del algoritmo. Es decir, se trata de aplicar una estrategia de simplificación del contexto desde un primer momento y al problema global, cosa que tiene mucha relación con la estrategia que se sigue a la hora de programar recursivamente. Esta opción ha sido la elegida en la aportación de este capítulo. La estrategia de disminución de la complejidad del contexto consistirá, en nuestro caso, en decrementar el número de genes del individuo implicado en las redes Bayesianas asociadas a las etapas más costosas del EDA.

La mayor parte de los esfuerzos en la mejora de la eficiencia se han basado en estrategias de paralelización, y la idea de aplicar ésta a algoritmos evolutivos no es nueva, como podemos ver en (4; 5; 38), ya que en principio este tipo de algoritmos es inmediatamente paralelizable, principalmente en dos sentidos. Comentémoslos brevemente aplicados a los algoritmos genéticos, que es el caso de (38). En el caso de los *algoritmos genéticos maestro-esclavo*, hay una única población y un proceso maestro que realiza las operaciones de selección, cruce y mutación, mientras que el cálculo de la función objetivo para los cromosomas de la población se distribuye entre distintos esclavos (a cada esclavo le toca evaluar sólo un subconjunto de cromosomas). Esto fuerza a un mínimo de dos comunicaciones por generación entre el maestro y cada esclavo: una para transmitir los cromosomas al esclavo y otra para recoger sus evaluaciones. Esto limita el número de esclavos a utilizar, ya que si éste es alto al final se pierde demasiado tiempo en las comunicaciones. La otra aproximación inmediata, los *algoritmos genéticos de grano grueso*, buscan alcanzar la convergencia más rápidamente usando diferentes poblaciones y haciendo por tanto de cada proceso algo más independiente, disminuyendo así las necesidades de comunicación que lastraban la aproximación anterior. Las distintas poblaciones evolucionan cada una por su lado hasta que determinado evento ocurre (este evento puede ser un tiempo preestablecido, un número de generaciones transcurridas, la convergencia de las poblaciones, etc). En ese momento, los mejores cromosomas de cada población *emigran* a poblaciones vecinas sustituyendo a otros cromosomas (que pueden ser los peores de esas poblaciones o no). Esta emigración tiene lugar para evitar los temidos máximos o mínimos locales. La vecindad de las diferentes poblaciones la da la topología del algoritmo genético paralelo. Cuanto mayor sea el grado de la topología (el número de vecinos de cada población) y el ratio de emigración, más tiempo se pierde en la comunicación entre los distintos procesos. En ambas aproximaciones, hay que tener mucho cuidado con los tamaños de las poblaciones, pues afectan mucho al resultado final y dependen de cada problema, lo cual es una de las principales desventajas de los algoritmos genéticos. En ninguno de los dos métodos comentados se realiza una simplificación

del contexto. Estos métodos son directamente trasladables a los EDA, pero hay otra aproximación algo más interesante: buscar los métodos de paralelización sobre fases intrínsecas del EDA, tal y como hemos comentado más arriba.

Hay muchas maneras de paralelizar un algoritmo. Algunas de ellas son incluso automáticas: muchos compiladores tienen ya opciones para intentar paralelizar determinados bucles, aunque la solución a la que llegan no puede compararse a una paralelización *desde cero*. No se pretende en este trabajo realizar una taxonomía de los distintos esquemas de comunicación entre procesos, o de las distintas maneras de asociar procesos a procesadores (véase (62) para una descripción detallada), por lo que sólo comentaremos que en todos los trabajos que se citan a continuación el esquema utilizado es el de maestro-esclavo, ya citado de soslayo cuando hablábamos de paralelización de algoritmos genéticos [3]. En este esquema, hay un proceso principal, el *maestro*, que se encarga de ejecutar las partes no paralelas del problema y de crear otros procesos, los *esclavos*, que resuelven en paralelo la parte paralelizable del mismo. Cada esclavo contribuye con su solución parcial a la solución global, que es calculada por el proceso maestro una vez ha recibido todas las soluciones parciales de los esclavos (ver Figura 7.1). Este proceso puede repetirse varias veces a lo largo de la ejecución del algoritmo entero. Desgraciadamente en esta estrategia el cuello de botella suele trasladarse al proceso maestro, ya que tiene que recibir la información de todos los esclavos y procesarla de manera secuencial, amén de preparar la información a enviar a los esclavos y de realizar sus propios calculos secuenciales no paralelizables. Es decir, el flujo del programa es paralelo solamente en determinados momentos. Por el contrario es relativamente sencillo de implementar y, sobretodo, de adecuar a un hardware concreto.

Aunque la estrategia de aumento de eficiencia que se va a afrontar en este capítulo, los EDA recursivos, es serial y no paralela, tal y como hemos comentado se presta a la paralelización, con lo que obtendríamos una mejora adicional. El esquema de comunicación más adecuado para la paralelización de la estrategia recursiva, que comentaremos brevemente al final del capítulo, no será sin embargo el de maestro-esclavo, sino el de *divide y vencerás*, con una división dicotómica de la complejidad del contexto. Este esquema, que permite una paralelización más intensa, es sin embargo complicado de adecuar al hardware, como veremos en su momento. Son las fases de la paralelización llamadas de *aglomeración* y *mapeo* (62) las que se complican, pero volveremos sobre este tema más adelante. A continuación analizaremos, pues, las estrategias de paralelización que se han llevado a cabo en otros trabajos.

[3] En general, es el esquema de comunicación más utilizado a la hora de paralelizar problemas.

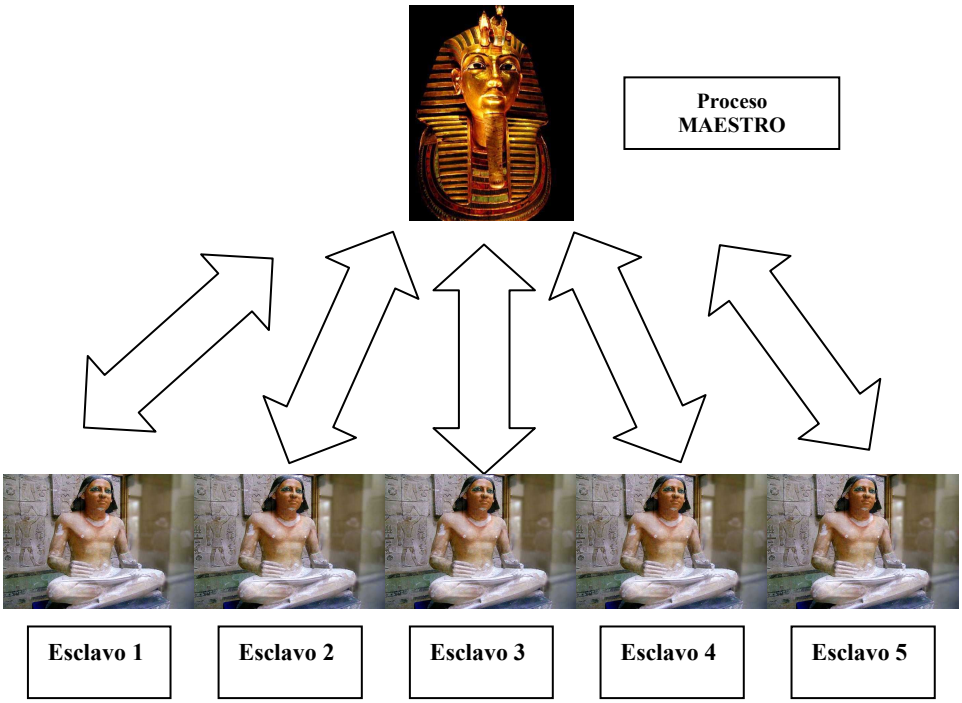

Figura 7.1: Esquema de comunicación maestro-esclavo.

7.1.1 Mejoras en el aprendizaje de la distribución

La táctica más inmediata puede ser restringir el tipo de parentesco de las variables de la red Bayesiana a la hora del aprendizaje. Sin embargo, en la mayor parte de los problemas, son precisamente los modelos no restrictivos como EBNA o EGNA (92) los que mejores resultados dan. Hay varios trabajos de paralelización que tratan de no aplicar restricciones en el modelo aprendido en cada generación y mantener así el comportamiento del EDA serial pudiendo aplicar la versión paralela a problemas más complejos para los que la versión serial es prohibitiva. Aunque todas estas técnicas, como las que aparecen en (17; 100; 105), se aplican a EDAs concretos ($EBNA_{BIC}$, $EBNA_{PC}$, $EGNA_{BIC}$ y $EGNA_{EE}$), se pueden trasladar a cualquier tipo de EDA.

Normalmente, los EDA utilizan una estrategia de búsqueda de la mejor red Bayesiana para la población en curso basada en métricas. En (17; 105), la métrica utilizada en el EDA $EBNA_{BIC}$ es la llamada *Bayesian information criterion* o BIC (134), que asocia una evaluación a cada red Bayesiana, consistiendo la estrategia de

búsqueda en el borrado o adición arcos. La idea consiste en descomponer la métrica original en varios sumandos en este caso, de manera que cada proceso se encargue de calcular cada uno de estos sumandos. La suma final se lleva a cabo en el proceso principal del programa paralelo, que es el que crea y controla el resto de los procesos. En el caso de la métrica BIC, que tiene la siguiente expresión (el significado de los distintos miembros de la ecuación pueden ser consultados en el capítulo 2):

$$BIC(S,D) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk} log \frac{N_{ijk}}{N_{ij}} - \frac{1}{2} logN \sum_{i=1}^{n} q_i(r_i - 1) \tag{7.1}$$

podemos reescribirla de esta manera:

$$BIC(S,D) = \sum_{i=1}^{n} BIC(i,S,D) \tag{7.2}$$

donde

$$BIC(i,S,D) = \sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk} log \frac{N_{ijk}}{N_{ij}} - \frac{1}{2} logN \sum_{i=1}^{n} q_i(r_i - 1) \tag{7.3}$$

así, cada proceso puede encargarse de un conjunto de variables a la hora de calcular la métrica o a la hora de analizar cómo influye un determinado cambio de estructura relacionado con su conjunto de variables en la métrica global. Esta distribución del trabajo provoca una mayor velocidad de la fase de aprendizaje. Sin embargo no podríamos aplicar esta mejora a EDAs que utilicen métricas no descomponibles. También puede ocurrir que la métrica sea poco descomponible y lo que ganemos con su paralelización lo perdamos en la creación y control de los distintos procesos. Y en cualquier caso, si cambia la métrica, hemos de ponernos a trabajar de nuevo para intentar descomponerla.

En (114) la métrica a utilizar para el algoritmo BOA (Bayesian Optimization Algorithm) es la Bayes-Dirichlet (76), que es también descomponible, aunque esta vez en factores de un producto en el que cada factor i esta relacionado con los arcos que acaban en la variable X_i. La paralelización hace que el tiempo de construcción de la mejor red Bayesiana (al fin y al cabo es un problema de aprendizaje lo que tenemos aquí) pase de $O(n^3)$ a $O(n^2)$.

7.1.2 Mejoras en la simulación de la red

En EDAs como $EBNA_{BIC}$, la fase de aprendizaje de la red es muy costosa, pero en otros, como $EGNA_{EE}$, también lo es la fase de simulación y evaluación de los diferentes individuos (véase (92) para una descripción de estos EDAs). Dado que el PLS, el algoritmo utilizado para simular los individuos en (105; 114) y en este trabajo, es secuencial (ver el capítulo 3), la única posibilidad de paralelización para la fase de simulación consiste en hacer que cada proceso se encargue de simular un conjunto de individuos (tantos como el tamaño de población dividido entre el número de procesadores). Es la opción que también se usa en (17), parecida a la paralelización de la evaluación de los cromosomas que se hacía en los genéticos maestro-esclavo.

7.1.3 Influencia de la arquitectura del hardware

Como ya hemos comentado, la información que transmiten los distintos procesos al proceso maestro es importante, porque si es considerable, se ralentiza la ejecución. Una manera de solventar un poco esto es ejecutar el programa en máquinas de memoria compartida usando threads (33). En estas máquinas todos los procesadores comparten la memoria RAM, esto es, la memoria es común a todos, lo que hace que el intercambio de la información no salga de la propia máquina. El gasto de tiempo empleado en transmitir la información por un switch se elimina (ver Figura 7.2). Sin embargo, las máquinas de memoria compartida son caras. Si minimizamos el trasvase de información entre procesos, podemos hacer el programa adecuado para ser ejecutado en un cluster, esto es, diferentes máquinas unidas mediante una red (que dependiendo de la tecnologa puede ser Ethernet, Myrinet, Infiniband, Quadrics) (ver Figura 7.3), cada una con su propia RAM, en la que la información viaja entre procesos a través de una red. En este caso se utilizan librerías de paso de mensajes que implican varios nodos, como MPI (*Message Passing Interface*) (61) o PVM (*Parallel Virtual Machine*) (72). Si el paso de mensajes se minimiza mucho, una tecnología muy barata como es hoy en día la Ethernet con switches de estado de latencia bajos puede ser suficiente. En concreto los clusters Beowulf (47), muy populares hoy en día y cuyo software es gratuíto desde el sistema operativo (Linux), los compiladores (GNU) que soportan programación multi-hilo (threads), hasta las librerías matemáticas ya paralelizadas (Blacs, Scalapack,...) y de comunicaciones (OpenMPI, Mpich, LAM-Mpi,...), son una opción muy válida. Nuestra estrategia recursiva no es como tal paralela, pero se presta a una paralelización *global* que se describe en la Sección 7.6. En resumen, si conseguimos minimizar el paso de información entre procesos, podremos ejecutar el EDA paralelo en máquinas muy baratas.

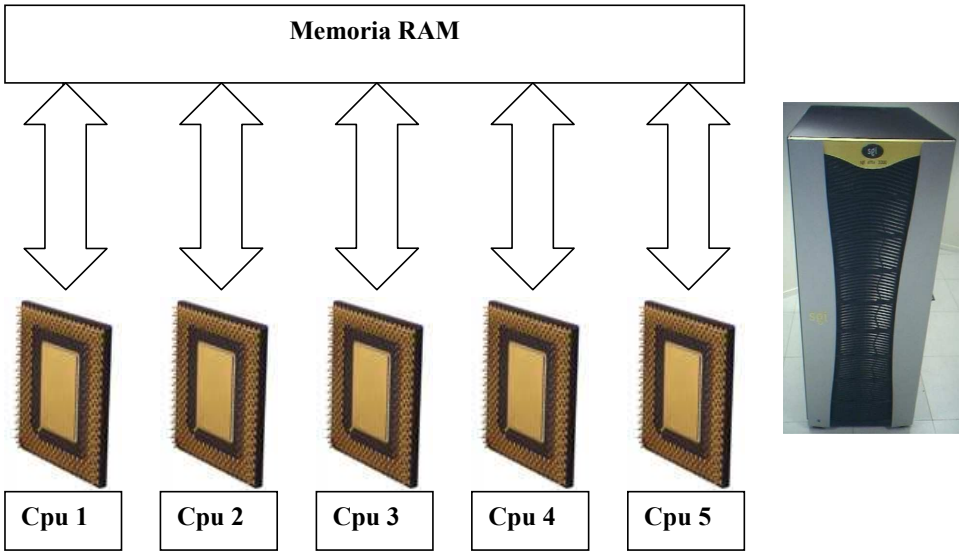

Figura 7.2: Máquina de memoria compartida. El intercambio de información entre CPUs sólo pasa por la memoria RAM (comunicación más rápida).

7.2 La recursividad

Los algoritmos recursivos son adecuados para aquellos problemas cuya solución puede plantearse como una combinación de las soluciones de otros problemas de la misma naturaleza pero de menor tamaño. Este *menor tamaño* es en el sentido de lo que se llama un orden bien fundado, esto es, que no haya cadenas descendentes infinitas. Más adelante nos referiremos a este menor tamaño con la expresión *simplificación del contexto*.

La recursividad es una estrategia del programador a la hora de resolver una dificultad algorítmica que consiste en dividir el problema inicial en distintos subproblemas más pequeños, y éstos últimos en otros más, y así sucesivamente, hasta llegar a problemas cuyo tamaño determina una solución sencilla. No vamos a tratar aquí de realizar una transformación de los algoritmos EDA propuestos para la triangulación, esto es, no obtendremos un algoritmo equivalente, sino uno que, sin traicionar la sólida base matemática que se encuentra en los cimientos de un EDA, consiga focalizar su ejecución en distintas partes del orden en cada invocación, de manera que tras la conclusión de todas las invocaciones obtengamos mejores resultados que la versión no recursiva, y además con un coste de tiempo menor.

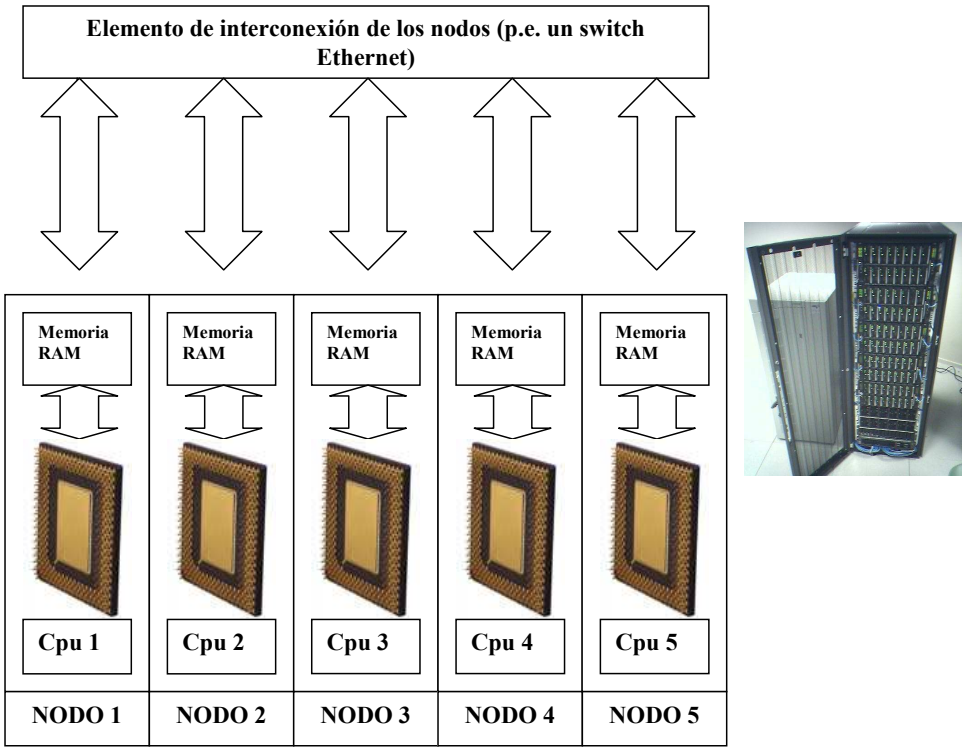

Figura 7.3: Máquina de memoria distribuida. El intercambio de información entre CPUs pasa por el elemento de interconexión de los nodos (comunicación más lenta).

7.2.1 Subprogramas

Cuando un analista informático afronta la solución de un problema, normalmente no intenta hacerlo con todo detalle desde un primer momento, sino que divide el problema global en subproblemas, resuelve cada subproblema de manera independiente prestando especial atención al intercambio de información entre el flujo global del programa principal y el subproblema en cuestión, y finalmente reune las *soluciones parciales* de los distintos subproblemas para conseguir la solución global que busca. A menudo, los subproblemas son resueltos mediante subprogramas, trozos de código independientes que reciben una serie de datos del programa principal y devuelven otros en los que queda plasmada la solución al subproblema.

Convertir la solución de un subproblema en un subprograma tiene sentido si el subprograma va a ser usado varias veces a lo largo de la ejecución del programa principal, o si queremos que nuestro código sea especialmente claro y legible, ya

que cada llamada a un subprograma implica toda una serie de transferencias de información entre el programa principal y el subprograma en ambos sentidos, y eso tiene siempre un coste en el tiempo de ejecución total. Por otra parte, un determinado subproblema puede llegar a ser lo suficientemente complejo como para que merezca la pena dividirlo a su vez en más subproblemas.

Factorial (N)
  $R \leftarrow 1$
  Para I *de* 1 *a* N *hacer*
    $R \leftarrow R \cdot I$
  FinPara
  Devolver R

Figura 7.4: Pseudocódigo iterativo para calcular un factorial.

Hay otro posible uso de los subprogramas que permite soluciones elegantes a problemas complejos: la ya mencionada recursión. Informalmente, decimos que un programa es recursivo cuando se invoca a sí mismo. Veamos un ejemplo. Intentemos confeccionar un subprograma no recursivo que devuelva el factorial de un número N. La solución la tenemos en la Figura 7.4. Si pensamos en la definición clásica del factorial de un número, el algoritmo es evidente, así como los parámetros que ha de recibir desde el flujo principal del programa para llevar a término su cometido. Sin embargo, podemos definir el factorial de otra manera, de manera *recursiva*:

$$Factorial(N) = Factorial(N-1) \cdot N \tag{7.4}$$

Esta manera de programar es trasladable directamente a la mayor parte de los lenguajes de programación actuales.

Hay que tener presente que la mayor parte de los programas recursivos se pueden traducir a programas iterativos, que siempre son bastante más rápidos, debido a que la llamada a un subprograma implica un trasvase de información del programa que llama al llamado y viceversa. Hay además un almacenamiento de información en memoria de los parámetros (ver siguiente sección) con cada llamada recursiva, y con frecuencia se repiten los cálculos al realizarse automáticamente llamadas con los mismos valores de los parámetros. Hay métodos sistemáticos de transformación de

programas recursivos en iterativos, como el método de Burstall o el de la inmersión. Y siempre se puede implementar una pila y transformar el programa en iterativo tal y como lo hace, de hecho, un compilador. En cualquier caso, la potencia de los ordenadores actuales y la posibilidad que tienen ya todos los lenguajes de realizar llamadas recursivas de manera eficiente hace innecesario, por lo general, esta transformación. Para una consulta detallada de estos métodos y, en general, de las decisiones implicadas en el diseño de cualquier algoritmo, (10; 11; 56; 79) pueden servir como bibliografía básica.

7.2.2 Diseño de un algoritmo recursivo

En un algoritmo recursivo es necesario determinar dos contextos de ejecución bien diferenciados, el de los llamados *casos básicos* y el de los *casos generales*. Veámoslo con un ejemplo ya clásico en la literatura sobre recursión como es el de las *Torres de Hanoi* (148).

El problema de las Torres de Hanoi consiste en lo siguiente: tenemos tres varillas, en la primera de las cuales se encuentran n discos concéntricos de diámetro creciente según se van acercando a la base de la varilla. En el resto de las varillas no hay discos. Se trata de trasladar todos los discos de la varilla 1 a la varilla 3 sin permitir que un disco de diámetro mayor se encuentre en ningún momento sobre un disco de diámetro menor.

Concebir desde un primer momento un algoritmo iterativo que resuelva el problema, esto es, que nos dé la secuencia de movimientos de discos que lleva a resolverlo, no es fácil. Es más sencillo, sin embargo, imaginar una solución de este tipo: podemos mover primero $n-1$ discos de la varilla 1 a la 2, luego el disco que queda en la varilla 1 a la 3, y a continuación los $n-1$ discos de la varilla 2 a la 3. Es decir, necesitamos un procedimiento que sea capaz de mover d discos de una varilla i a una varilla j: este será nuestro procedimiento recursivo, que llamaremos, por ejemplo, $Hanoi(d, i, j)$. Así, nuestro problema inicial, $Hanoi(n, 1, 3)$ puede resolverse haciendo tres simples llamadas recursivas:

- $Hanoi(n-1, 1, 2)$
- $Hanoi(1, 1, 3)$
- $Hanoi(n-1, 2, 3)$

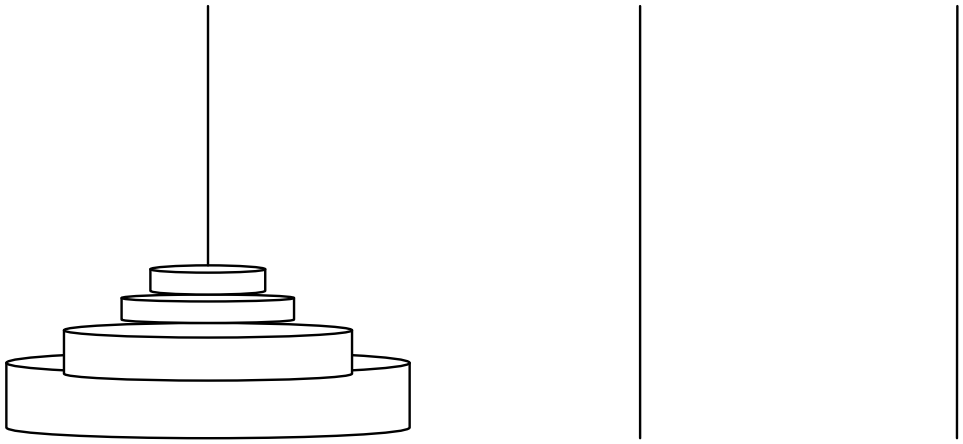

Figura 7.5: Estado inicial del problema de las Torres de Hanoi.

Hemos identificado aquí el *caso general*, aquel en el que se realiza la llamada recursiva. Sin embargo, hemos de tener en cuenta que la línea $Hanoi(n-1,1,2)$ implica a su vez otra serie de llamadas recursivas para resolver ese subproblema:

- $Hanoi(n-2,1,3)$
- $Hanoi(1,1,2)$
- $Hanoi(n-2,3,2)$

Como vemos, si cada llamada a $Hanoi$ genera siempre otras tres llamadas, el algoritmo no termina nunca. Tiene que haber, evidentemente, un caso que no implique una llamada recursiva y que llamaremos *caso básico* de la recursión (puede haber más de uno). En el ejemplo que nos ocupa, aparece cuando el número de discos que hay que mover es solamente uno. Podemos ver una descripción más gráfica del problema y el pseudocódigo de $Hanoi$ en las Figuras 7.5 y 7.6.

Ya podemos adelantar que la estrategia que seguiremos en los EDA recursivos consistirá en *fijar* la mitad de los nodos del orden (es decir, estos nodos no van a participar en la evolución) y llamar recursivamente al EDA para que optimice el orden de los nodos no fijados. Después, fijaremos la otra mitad de los nodos y volveremos a llamar al EDA para que evolucione sobre ellos. Como vemos, las poblaciones en las llamadas recursivas contienen individuos cuyo número de genes es la mitad de

```
Procedimiento Hanoi(d,i,j)
    Si d = 1 entonces Mover disco de i a j
    Sino
        k ← 6 − i − j
        Hanoi(d − 1, i, k)
        Hanoi(1, i, j)
        Hanoi(d − 1, k, j)
    FinSi
```

Figura 7.6: Pseudocódigo recursivo para solucionar el problema de las Torres de Hanoi.

los de la invocación que las llama [4]. Según avanza la recursión, el número de nodos fijados es cada vez mayor, y por tanto el número de genes de los individuos de las poblaciones es menor, y la velocidad de ejecución del EDA se incrementa.

7.3 El EDA recursivo

En general, en el diseño de un programa recursivo hablamos de seis etapas:

- La fase de *parametrización*, en la que tratamos de encontrar los parámetros mediante los cuales el problema va a poder dividirse en subproblemas de tamaño más pequeño. Podemos ya adelantar que en nuestro caso el parámetro principal va a ser el conjunto de nodos $G_r \subset G$ que no están fijados en el nivel de recursión r, es decir, el conjunto de nodos cuyo orden relativo variará según el EDA va proponiendo individuos. Además, enviaremos como parámetro también el tamaño m_r del conjunto G_r. Es evidente que el tamaño del problema decrece según un orden bien fundado.

[4]Esto es sólo cierto en el caso de los EDA continuos, ya que en los discretos no hay una relación 1 a 1 entre el conjunto de nodos y el conjunto de genes del individuo EDA, aunque es evidente que si el número de nodos que participan en la optimización parcial del orden es menor, lo será también el número de genes del individuo usado para representar ese orden. Este comentario sólo tiene sentido si hablamos de búsquedas de permutaciones por la inusual relación que hay entre el individuo discreto que evoluciona en el EDA y la permutación que necesitamos para realizar la evaluación.

- La fase de *análisis de casos triviales*. Aquí se identifican los valores para los cuales el problema se puede resolver de manera directa y no recurrente, y se explicitan cuáles son estas soluciones. En nuestro caso, no tiene mucho sentido empezar a hacer llamadas recursivas cuando el número de nodos que no están fijados es menor que cuatro, ya que dada la simplificación dicotómica que realizamos, acabaríamos haciendo una llamada recursiva para un único nodo. Basta en este caso con resolver el problema utilizando el EDA secuencial normal que hemos venido usando hasta ahora y que con un tamaño de individuo tan pequeño no debería normalmente tardar mucho.

- La fase de *análisis de casos generales*. Aquí identificamos los valores para los que la solución es recurrente (número de nodos no fijados mayor o igual que cuatro en nuestro caso) y la combinación de llamadas recursivas con la que podemos hallarla (las dos llamadas ya descritas informalmente más arriba).

- La fase de *formulación del algoritmo*. En esta fase hay que describir claramente que es lo que tiene que hacer el algoritmo con sus parámetros, cosa que hemos definido informalmente más arriba y que se explicita más adelante en la Sección 7.3. Aquí intentamos no repetir innecesariamente determinados cálculos, eliminar casos triviales que pudieran sobrar, confirmar si la descomposición en casos es la adecuada (los problemas no tenidos en cuenta en las fases anteriores aparecen normalmente cuando se intenta describir el algoritmo con detalle), etc.

- La fase de *análisis de la corrección*, en la que vamos *siguiendo* el algoritmo paso a paso con algún ejemplo sencillo para confirmar que devuelve la solución correcta y si las llamadas se producen en el orden que queremos de tal manera que el programa termine. Un problema típico de este tipo de algoritmos es una secuencia de llamadas infinita por no visitar nunca uno de los casos triviales, que acaba finalmente con un error en tiempo de ejecución por el agotamiento de los recursos de la máquina. Siendo rigurosos, la terminación ha de probarse por inducción. En el paso básico, es evidente que el programa termina porque no realiza ninguna llamada recursiva. Y en el paso de inducción, donde hemos de comprobar que la llamada para G_r termina (donde r es el nivel de recursión), hemos de realizar la siguiente hipótesis de inducción: $\forall G_k \subset G(4 \leq m_k \leq m_r \Rightarrow$ *la llamada para* G_k *termina*). La llamada para G_r produce dos llamadas más que las llamadas para G_{r+1}, y en ambas llamadas $m_{r+1} < m_r$ puesto que lo que se hace es fijar más nodos. Por tanto, el algoritmo termina. La

demostración por inducción de la corrección del algoritmo es también evidente, dado cómo se define el algoritmo en las Figuras 7.8, 7.9 y 7.10. En este caso la hipótesis de inducción sería: $\forall G_k \subset G(4 \leq m_k \leq m_r \Rightarrow$ *El algoritmo devuelve el conjunto de órdenes con mejor evaluación que el EDA encuentra durante su ejecución desde el nivel de recursión número 0 hasta el nivel de recursión número* k*, incluido éste último*). Este conjunto de individuos no tiene porqué ser el mejor en sentido absoluto, dado el caracter no determinista de los algoritmos evolutivos en general.

- La fase de *implementación*, en la que se toman decisiones acerca de las estructuras o tipos de datos a utilizar y las acciones concretas, que tiene ya mucha relación con el tipo de lenguaje de programación que vamos a utilizar.

En la descripción del pseudocódigo del algoritmo recursivo utilizado en los experimentos se hace referencia a una serie de conceptos que pasamos a describir a continuación:

- ζ^{α} es una cache de órdenes completos de m nodos. Esta cache se mantiene actualizada a lo largo de la ejecución de todo el programa con los M mejores órdenes (esto es, se guardan tantos órdenes como el tamaño de población), y es una cache *ordenada* según el valor de la evaluación de los distintos órdenes. Como veremos más adelante, utilizaremos esta cache para realizar la evaluación de los individuos y para inicializar las poblaciones.

- G es el conjunto de m nodos de la red Bayesiana que tratamos de triangular.

- $G_r \subset G$ es el subconjunto de m_r nodos de G que no están fijados en el nivel de recursión r.

- Llamaremos *nivel de recursión* r al número de sucesivas llamadas recursivas encadenadas ya realizadas, esto es, a la profundidad de la recursión. Así, el nivel de recursión en el programa principal es cero, en la primera llamada recursiva es 1, en la siguiente llamada recursiva realizada por la anterior r pasa a ser 2, y así sucesivamente.

- g_{m_r} es el número de genes que tiene el individuo que representa un orden de m_r nodos. En los dominios continuos, $m_r = g_{m_r}$, pero no es así en los discretos, en los que no hay una relación 1 a 1 entre los nodos y los genes. g_m es el número de genes que tiene un individuo que representa un orden completo con los m nodos de G.

- D_l es una población de individuos de $g_{m_r} \leq g_m$ genes en el nivel de recursión r y en la generación l. Aquí podríamos haber utilizado la notación D_l^r, esto es, incorporando el nivel de recursión a la población, pero hemos decidido no hacerlo para hacer la notación más sencilla, ya que esta variable es siempre local en los pseudocódigos. Hemos procedido de la misma manera siempre que ha sido posible. Téngase en cuenta de nuevo que el número de genes g_{m_r} de un individuo no es igual al número de nodos que no se encuentran fijados en ese nivel de recursión (esto último solo ocurre en los dominios continuos).

- c_i^α es el $i-$ésimo orden de m nodos perteneciente a la cache ζ^α. Dada la definición de la cache, c_0^α será siempre el mejor orden encontrado hasta el momento.

- c_i^β es un orden de $m_r \leq m$ nodos *extraidos* de un c_i^α, compuesto por los nodos del segundo que no están fijados en el nivel de recursión r, respetando el orden relativo que tienen estos nodos en c_i^α. Como hemos comentado antes brevemente, fijaremos los nodos para indicar al algoritmo sobre qué nodos debe evolucionar y sobre cuáles no (los fijados). De esta manera estamos evolucionando sobre subconjuntos compuestos por nodos cuyos índices no tienen porqué ser consecutivos, sino que se seleccionarán aleatoriamente.

- i^α es un orden cualquiera de m nodos. c^α es un i^α que pertenece a la cache.

- i_{ind}^α es el individuo de g_m genes que está asociado al orden i^α.

- i^β es un orden cualquiera de $m_r \leq m$ nodos, todos ellos pertenecientes a G. c^β es un i^β extraido de un c^α.

- i_{ind}^β es el individuo de g_{m_r} genes asociado al orden i^β. Cuando hablemos de *individuo* β nos estaremos refiriendo a este tipo de individuo. Todo i_{ind}^β tiene un i^β asociado y al revés, aunque sólo en el caso de los dominios discretos esta correspondencia es biunivoca, dada la representación del individuo utilizada en este trabajo.

- i es un orden cualquiera (de tipo α, β o γ [5]).

- $Nodo_j(i)$ es el nodo del orden i que está en la posición número j.

[5]Este último se describe más adelante.

- $G_r \subset G$ el subconjunto de m_r nodos G que no están fijados en el nivel de recursión r, esto es, los que aparecen en los órdenes i^β y sobre los que evoluciona el EDA en este nivel de recursión.

- i^γ es un orden de m nodos construido a partir de un i^β. Se construye colocando en su lugar los nodos fijados en el nivel de recursión r, y colocando a continuación los nodos que se encuentran en i^β en los huecos que quedan, respetando su orden relativo [6]. Evidentemente, la métrica o evaluación del orden i^β se calculará en tanto que convertido a su correspondiente i^γ.

- $Evaluar^\alpha(i^\alpha)$ es una función que devuelve la evaluación de un orden de m nodos.

- $Evaluar^\beta(i^\beta, i^\gamma)$ es un procedimiento que construye el orden de m nodos i^γ a partir del orden de m_r nodos, i^β, y devuelve $Evaluar^\alpha(i^\gamma)$, además del propio i^γ.

En las Figuras 7.8, 7.9 y 7.10 podemos ver el pseudocódigo utilizado en la estrategia recursiva, sensiblemente diferente al usado en los otros capítulos de este trabajo. Como vemos, es el *número de nodos no fijados* el que va a determinar cuál es el caso básico de la recursión y cuál el caso general.

7.3.1 Inicialización de la población del EDA

Si dejáramos al EDA evolucionar en un determinado nivel de recursión partiendo siempre de una simple incialización aleatoria, no aprovecharíamos lo conseguido en las llamadas recursivas previas. La idea de los EDAs recursivos consiste en ir evolucionando sobre partes cada vez más pequeñas del individuo para *refinar* el resultado obtenido hasta el momento, por lo que se hace necesaria una cache que mantenga los mejores órdenes de m nodos obtenidos previamente. Evidentemente, la primera llamada del algoritmo sí tendrá que partir de una inicialización aleatoria, puesto que aún no hay cache.

Sin embargo, al incorporar en la población los mejores individuos/órdenes en la inicialización de un determinado nivel de recursión, sólo se pueden tener en cuenta los nodos que no han sido fijados en el paso anterior, esto es, los nodos que se

[6] El método de construcción de los i^γ finalmente utilizado será un poco más complejo, ya que intentaremos aprovechar información presente en el mejor de los órdenes encontrados hasta el momento, tal y como veremos más adelante en la Sección 7.3.2.

I. Mejor orden de la última cache (m=16 genes) en el primer nivel de recursión después de la primera ejecución del *EdaBasico*. En este nivel, los nodos 9, 3, 6,16,12,15,4 y 7 ya han sido fijados.															
5	**9**	2	**3**	**6**	**16**	13	**12**	14	11	**15**	1	**4**	8	**7**	10
II. Nodos seleccionados para participar en la evolución del EDA en la primera llamada recursiva (y que se fijarán en la segunda de este mismo nivel de recursión).															
2				*14*				*1*				*10*			
III. Nodos que se fijarán en la próxima llamada recursiva (y que participarán en la evolución en la segunda de este mismo nivel de recursión).															
5				**13**				**11**				**8**			
IV. Orden relativo de los nodos fijados durante la ejecución de la llamada recursiva en el nivel de recursión número dos.															
5	**9**	**3**	**6**	**16**	**13**	**12**	**11**	**15**	**4**	**8**	**7**				
V. Individuo/orden beta propuesto por el EDA.															
10				*14*				*1*				*2*			
VI. Individuo/orden gamma que es finalmente evaluado.															
5	**9**	*10*	**3**	**6**	**16**	**13**	**12**	*14*	**11**	**15**	*1*	**4**	**8**	**7**	*2*

Figura 7.7: Paso de un i^{β} a un i^{γ}.

van a tener en cuenta en la evolución. Esto hace que se incorpore a la población inicial la información acerca de las posiciones relativas de los nodos no fijados que se encuentran en los mejores órdenes encontrados, perdiendo los mejores órdenes como tales. Parece aconsejable que los mejores i^{β} encontrados en el nivel de recursión r no sustituyan completamente, una vez transformados en i^{γ}, a los que se encuentran en la cache ζ^{α}. Podrían ser añadidos a ella sólo si su evaluación es mejor que la del peor de los individuos de la cache, manteniendo ésta con un tamaño M (para poder así utilizarla como entrada de la inicialización del paso de recursión siguiente).

Sin embargo, esto puede hacer que el resultado de las distintas llamadas recursivas esté demasiado determinado por la primera ejecución, aquella que partió de una inicialización aleatoria. En los experimentos se introducirá una mayor variedad en las poblaciones manteniendo en la nueva ζ^{α} únicamente (por ejemplo) los cinco mejores órdenes (en vez de la mayor parte de ellos, en el caso de fuesen pocos los i^{β} que mejoraran sus evaluaciones), siendo el resto sustituidos por los mejores obtenidos en la invocación recursiva en curso. Pero esto no quiere decir que inicialicemos la población del EDA con sólo cinco individuos: la población inicial del EDA y la cache ζ^{α} son cosas diferentes. De hecho, la primera está compuesta de *individuos* de g_{m_r} genes, mientras que la segunda está compuesta de órdenes de m nodos. Para la inicialización utilizaremos todos los órdenes de la cache, esto es, en el procedimiento $InicializarPoblacionUsandoCache$ se usan todos los individuos de ζ^{α}, como se puede

Procedimiento **EDABasico** (m_r, G_r)
  Si $\zeta^{\alpha}_{viejo} \neq \varnothing$ *entonces*
    InicializarPoblacionUsandoCache (ζ^{α}_{viejo}, D_0, G_r, i^{α}_{mejor})
  Sino
    $D_0 \leftarrow$ Generar M individuos aleatoriamente
    $\zeta^{\alpha} \leftarrow \varnothing$
  FinSi
  $l \leftarrow 1$
  Repetir
    Para cada individuo $i^{\beta}_{ind} \in D_{l-1}$ *hacer*
      Evaluar$^{\beta}$(i^{α}_{mejor}, i^{β}, i^{γ}, s^{β})
      Si s^{β} es la mejor evaluación hasta el momento *entonces*
        insertar i^{γ} en ζ^{α} en el primer puesto
        Si $\|\zeta^{\alpha}\| > M$, quitar el individuo peor de ζ^{α} (el último)
      FinSi
    FinPara
    $D^{s}_{l-1} \leftarrow$ Seleccionar $N \leq M$ individuos de D_{l-1} utilizando un
        método de selección prefijado
    $\rho_l(\boldsymbol{z}) \leftarrow$ Estimar la distribución de probabilidad de $\boldsymbol{z}$ dado D^{s}_{l-1}
    $D_l \leftarrow$ Simular M individuos (la nueva población) a partir de $\rho_l(\boldsymbol{z})$
    $l \leftarrow l + 1$
  Hasta que un criterio de parada relacionado con las generaciones ya
      obtenidas en el paso recursivo sea satisfecho
  $\zeta^{\alpha}_{viejo} \leftarrow \zeta^{\alpha}$

Figura 7.8: Pseudocódigo de un bucle básico del EDA (no recursivo). Hay que tener en cuenta que aquí las poblaciones D_l están compuestas por individuos de m_r genes

ver en el pseudocódigo de la Figura 7.11. El lector se preguntará por el motivo por el cual el primer orden de la cache (que se copia a la variable i^{α}_{mejor}) se pasa como parámetro por valor al procedimiento *Evaluar*$^{\beta}$. Esto sirve para una mejora sustancial del algoritmo tal y como veremos en la próxima sección.

Procedimiento **EDARecursivo** (m_r, G_r)
Si $m_r < 4$ *entonces*
no hacer nada
Sino
EdaBasico (m_r, G_r)
Seleccionar $trunc(\frac{m_r}{2})$ nodos de G_r aleatoriamente.
Sea $G_{r+1} \subset G_r$ el conjunto de los $trunc(\frac{m_r}{2})$ nodos seleccionados.
$m_{r+1} \leftarrow trunc(\frac{m_r}{2})$
EdaRecursivo (m_{r+1}, G_{r+1})
$G'_{r+1} \leftarrow G_r \setminus G_{r+1}$
EdaRecursivo ($m_r - m_{r+1}$, G'_{r+1})
FinSi
EdaBasico (m_r, G_r)

Figura 7.9: Pseudocódigo recursivo del EDA propuesto. Comienza con una evolución sobre el (sub)conjunto de nodos completo para tener en cuenta en algún momento las interrelaciones entre todos los nodos del mismo. Si no lo hiciéramos así, sólo evolucionariamos en el caso básico. A continuación de las llamadas recursivas se realiza siempre otra llamada no recursiva para aprovechar la cache actualizada por las dos recursivas previas.

EDA
$\zeta^{\alpha}_{viejo} \leftarrow \varnothing$
$EdaRecursivo(m, G)$
Devolver c^{α}_0

Figura 7.10: Pseudocódigo principal del EDA.

7.3.2 La evaluación de los órdenes i^{β}

En la transformación de un orden i^{β} en otro i^{γ} propuesta en un primer momento, los nodos fijados se colocaban en su lugar correspondiente (es decir, el nodo 3 se coloca el tercero en el orden en el caso de que esté fijado, etc). Sin embargo, tras la primera

Procedimiento **InicializarPoblacionUsandoCache** (ζ^{α}_{viejo}, D, G_r, i^{α}_{mejor})
$i^{\alpha}_{mejor} \leftarrow c^{\alpha}_{viejo_0}$
$\zeta^{\alpha} \leftarrow \varnothing$
Para cada orden $c^{\alpha} \in \zeta^{\alpha}_{viejo}$ *hacer*
Si $i \leq 5$ *entonces*
$s^{\alpha} \leftarrow Evaluar^{\alpha}(c^{\alpha})$
Insertar c^{α} en ζ^{α} según el valor de s^{α}
FinSi
$Evaluar^{\beta}(i^{\alpha}_{mejor}, c^{\beta}, i^{\gamma}, s^{\beta})$
Insertar i^{γ} en ζ^{α} según el valor de s^{β}_{i}
Si $\|\zeta^{\alpha}\| > M$, quitar el individuo peor de ζ^{α} (el último)
Añadir c^{β}_{ind} a D
FinPara

Figura 7.11: Pseudocódigo del procedimiento $InicializarPoblacionUsandoCache$.

ejecución de $EdaBasico$ ya tenemos una cache con órdenes más o menos buenos cuya información podríamos quizás añadir a la hora de la evaluación: podríamos recurrir al primer orden de la cache, el de mejor evaluación (i^{α}_{mejor}) y utilizar *el orden relativo* que tienen los nodos fijados en ese orden. En la Figura 7.7 podemos ver un ejemplo de cómo se construye un i^{γ} a partir de un i^{β}. Con esto conseguimos dos cosas:

- *Enriquecer* los individuos de m_r genes propuestos por el EDA con la información del mejor individuo/órden del que disponemos hasta ese momento. No olvidemos que los i^{γ} se incorporan a la cache sin son buenos.

- Mezclar, en el momento de la carga de la cache, órdenes ya de por si *buenos* con el mejor, en lo que a los nodos fijados se refiere.

Esto es lo que se propone en el procedimiento $Evaluar^{\beta}$ cuyo pseudocódigo aparece en la Figura 7.12. Cabe la posibilidad de una mejora ulterior que no se ha utilizado en los experimentos: hacer que el valor de i^{α}_{mejor} sea dinámico dentro del propio procedimiento $EdaBasico$.

Procedimiento **Evaluar**$^{\beta}(i^{\alpha}_{mejor}, i^{\beta}, i^{\gamma}, s^{\beta})$
  $k \leftarrow 1$
  Para cada posición $j = 1, \ldots, m$ del orden i^{α}_{mejor} *hacer*
    Si $Nodo_j(i^{\alpha}_{mejor})$ está fijado *entonces*
      $Nodo_j(i^{\gamma}) \leftarrow Nodo_j(i^{\alpha}_{mejor})$
    Sino
      $Nodo_j(i^{\gamma}) \leftarrow Nodo_k(i^{\beta})$
      $k \leftarrow k + 1$
    FinSi
  FinPara
  $s^{\beta} \leftarrow Evaluar^{\alpha}(i^{\gamma})$

Figura 7.12: Pseudocódigo del procedimiento $Evaluar^{\beta}$.

7.3.3 Criterio de parada

Hay que mantener la producción de generaciones el tiempo suficiente para que por lo menos la cache se rellene con M individuos, cosa que a no ser que se repitan muchos individuos pasa normalmente una vez transcurrida la primera generación. ¿Cuánto tiempo hay que mantener la evolución? ¿Quizás debieran transcurrir más generaciones cuando el tamaño del individuo es grande y menos cuando no lo es? ¿Quizás fuese mejor lo contrario? Esta segunda opción haría que el EDA fuese más rápido, pues manejar individuos más pequeños es siempre menos costoso computacionalmente hablando a la hora de trabajar con la red Bayesiana del EDA, y esa es una de las grandes ventajas de la opción recursiva. En los experimentos se ha optado por una solución salomónica: en cada paso recursivo se produce exactamente la misma cantidad de generaciones. La cantidad concreta viene dada por el número de muestras con las que deseamos contar, que se *reparten* entre las distintas llamadas al *EdaBasico*.

¿Qué es lo que pasaría si, sobre la cache final que resta de la ejecución del EDA recursivo, volvieramos a ejecutar el EDA? ¿Refinaríamos razonablemente el resultado o por el contrario no obtendríamos una mejora acorde con el gasto computacional que representa? En las Figuras 7.13 y 7.14 podemos ver la influencia que tiene para la red Sparse, en poblaciones de 100 y 500 individuos, el hecho de ejecutar varias veces el EDA recursivo (en este caso la versión contínua de UMDA) sobre las caches

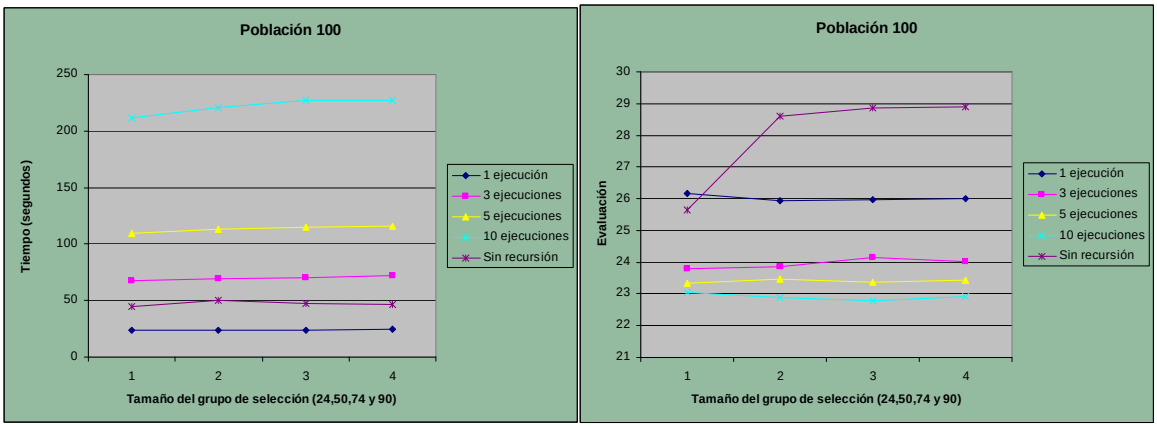

Figura 7.13: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 100 individuos, EDAs continuos, aprendizaje UMDA y tamaños del grupo de selección de 24, 50, 74 y 90 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

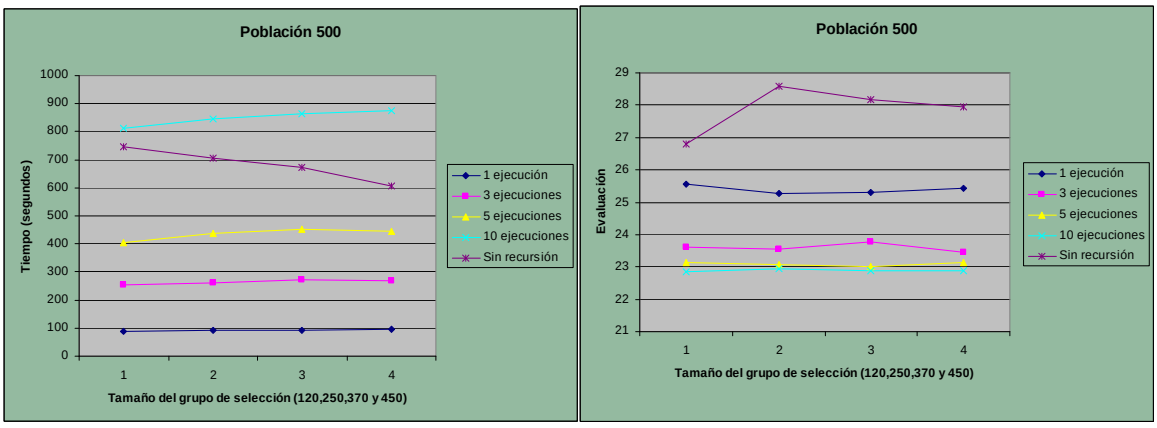

Figura 7.14: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 500 individuos, EDAs continuos, aprendizaje UMDA y tamaños del grupo de selección de 120, 250, 370 y 450 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

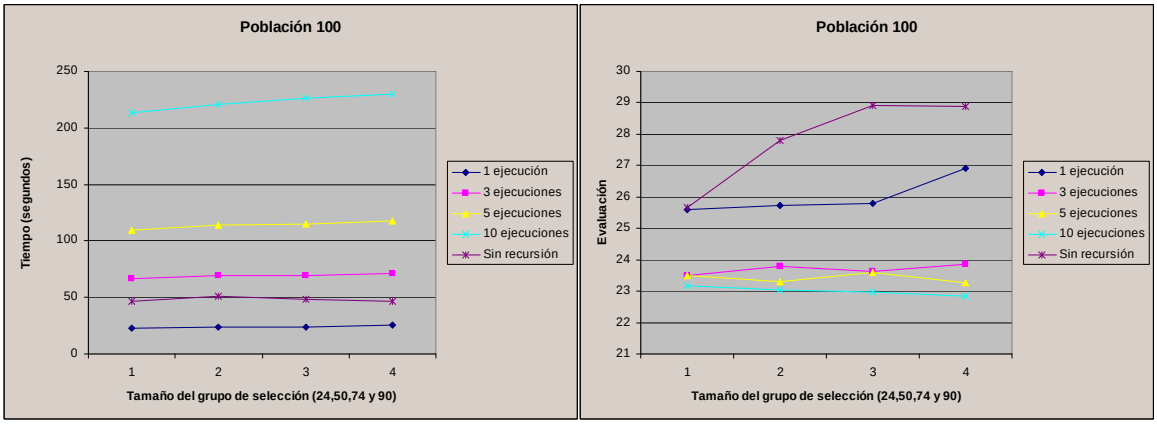

Figura 7.15: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 100 individuos, EDAs continuos, aprendizaje MIMIC y tamaños del grupo de selección de 24, 50, 74 y 90 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

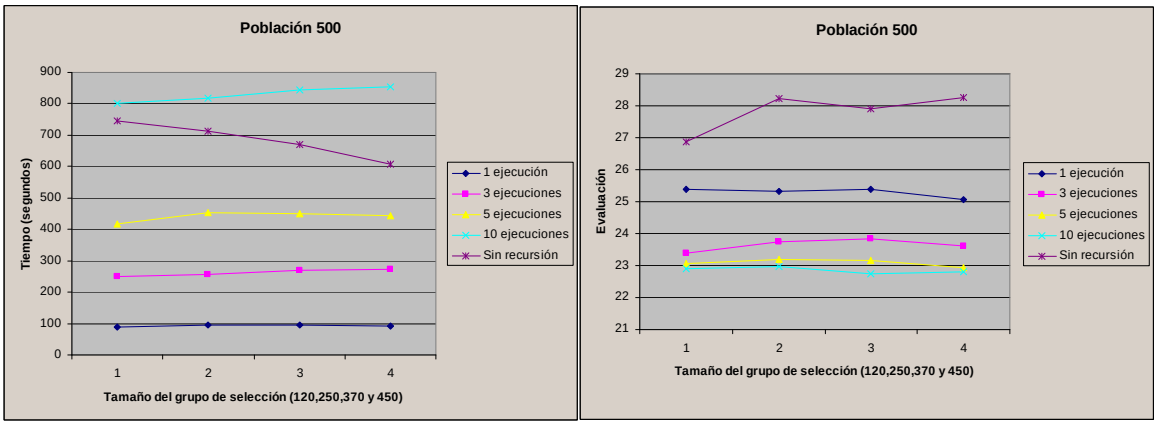

Figura 7.16: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 500 individuos, EDAs continuos, aprendizaje MIMIC y tamaños del grupo de selección de 120, 250, 370 y 450 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

que se van obteniendo, en las que las evaluaciones son cada vez mejores y los órdenes relativos entre nodos cada vez más cercanos a los que deberían tener los mejores individuos (cada valor es una media de 10 ejecuciones).

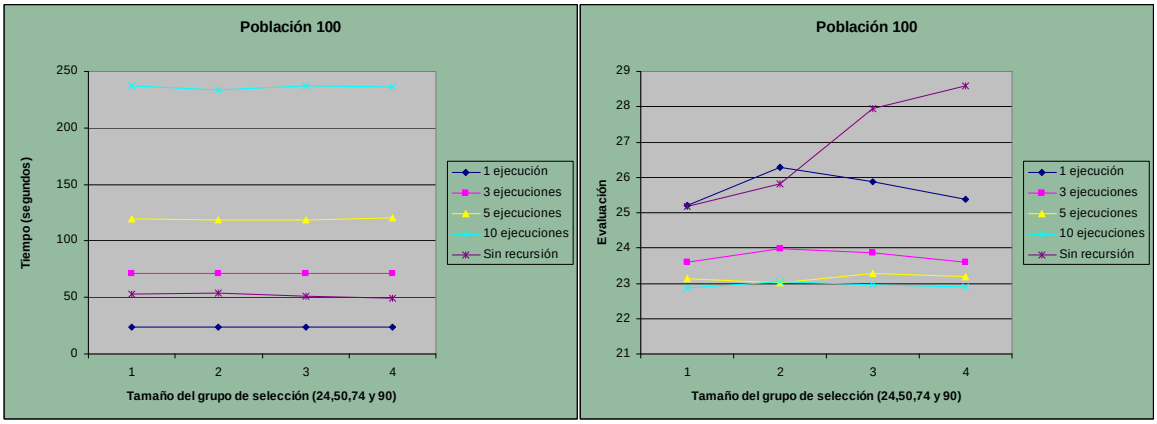

Figura 7.17: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 100 individuos, EDAs discretos, aprendizaje UMDA y tamaños del grupo de selección de 24, 50, 74 y 90 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

En el caso de las evaluaciones, se ve que el resultado final no deja de mejorar según vamos realizando ejecuciones del EDA continuo. Sin embargo, el tiempo también va aumentando, por lo que se trata de llegar a un compromiso entre buenos órdenes obtenidos y un tiempo de ejecución razonable. Parece que cinco ejecuciones dan un resultado bueno con un tiempo de ejecución no muy alejado de la ejecución sin recursión utilizada en el capítulo 4, aunque con una mejora aparente en los resultados apreciable que confirmaremos más tarde. Pero incluso una única ejecución del EDA recursivo ya parece ser mejor, excepto para determinados tamaños del grupo de selección, que el tradicional, tanto en tiempo de ejecución como en resultados. A este respecto, llama la atención lo sensible que es el EDA tradicional, en contraposición a la aparente estabilidad de los EDA recursivos, a los distintos tamaños del grupo de selección, tanto en lo que se refiere al tiempo de ejecución como a la evaluación del mejor individuo.

¿Obtendremos los mismos resultados con los otros tipos de aprendizaje y, lo más importante, con los EDA discretos? En las Figuras 7.15 y 7.16 podemos ver los resultados de los continuos para el EDA MIMIC. Vemos que las tendencias ya vistas con UMDA parecen confirmarse. En las Figuras 7.17 y 7.18, podemos ver los resultados para los EDA discretos con aprendizaje UMDA y en 7.19 y 7.20 los relativos al aprendizaje MIMIC. También observamos aquí una buena relación evaluación-tiempo de ejecución para las cinco ejecuciones consecutivas del EDA recursivo, así como una

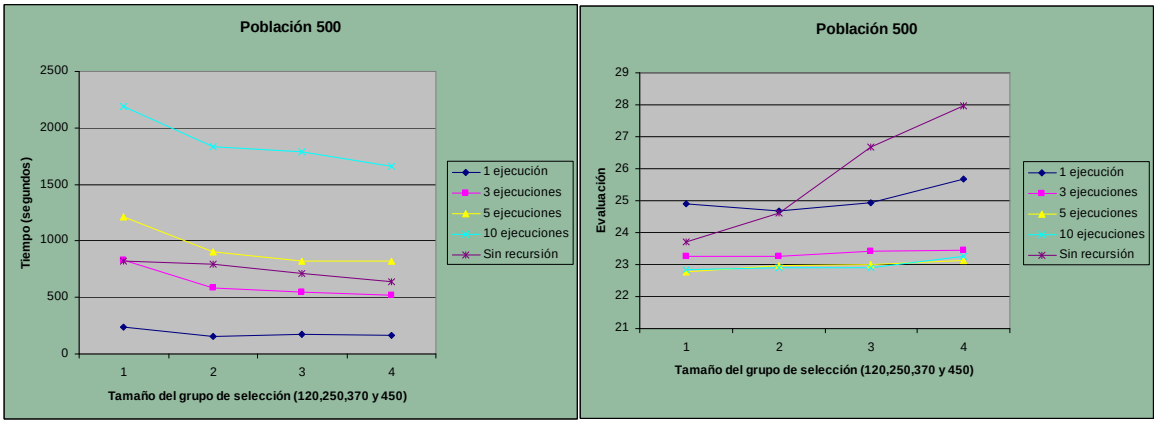

Figura 7.18: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 500 individuos, EDAs discretos, aprendizaje UMDA y tamaños del grupo de selección de 120, 250, 370 y 450 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

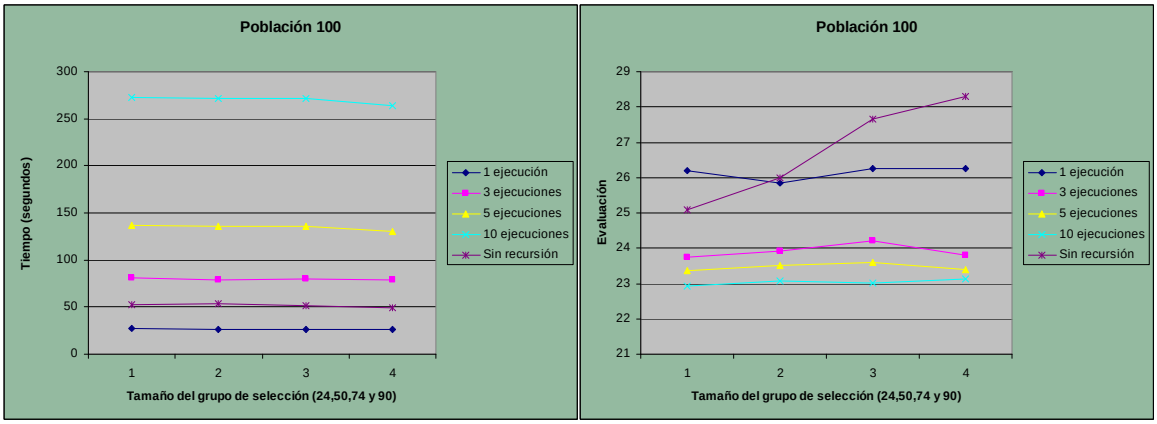

Figura 7.19: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 100 individuos, EDAs discretos, aprendizaje MIMIC y tamaños del grupo de selección de 24, 50, 74 y 90 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

interesante tendencia a la estabilidad con respecto al tamaño del grupo de individuos seleccionados para la generación de la red Bayesiana.

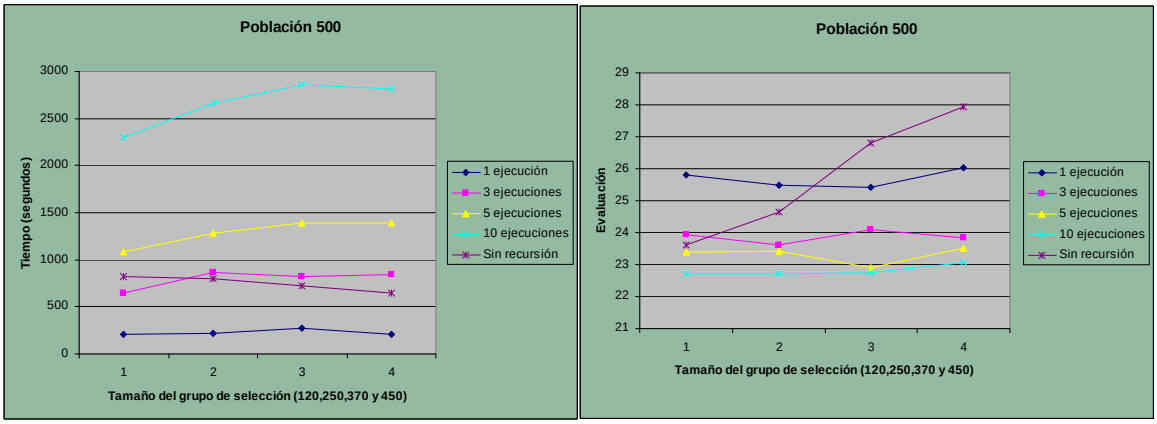

Figura 7.20: Evolución del tiempo de ejecución y de la mejor evaluación para poblaciones de 500 individuos, EDAs discretos, aprendizaje MIMIC y tamaños del grupo de selección de 120, 250, 370 y 450 individuos con diferente número de ejecuciones del EDA recursivo. Los valores son una media de 10 ejecuciones.

7.3.4 Parametrización del EDA

En las Figuras 7.13 a 7.20 podemos ver el efecto que tiene la variación del tamaño del grupo de selección tanto en el tiempo de ejecución como en la mejor evaluación obtenida para distinto número de ejecuciones del EDA recursivo. Los tiempos hay que utilizarlos con precaución, ya que los experimentos se han realizado en una máquina multiusuario en la que la disponibilidad de los recursos (porcentaje de CPU, memoria RAM,...) va variando dinámicamente. En general, el EDA recursivo, al contrario que el EDA tradicional, es bastante estable y da los mismos resultados (en principio mejores que el EDA tradicional) independientemente del tamaño del grupo de selección. Mientras que el tiempo de ejecución varía mucho con este tamaño en el EDA tradicional, las variaciones en el recursivo son pequeñas. En ocasiones, el EDA tradicional incluso arroja ejecuciones más costosas con grupos de selección pequeños que con otros más grandes en los que es de esperar un mayor coste computacional. Esto probablemente ocurre porque cuando el grupo de selección es pequeño, el EDA propone siempre los mismos órdenes, y recordemos que en nuestro algoritmo solo se realizan muestras (las cuales determinan el tiempo de ejecución) cada vez que se alcanzan cierto número de evaluaciones de individuos distintos y no propuestos anteriormente. Llama la atención el en principio mal comportamiento del EDA tradicional con tamaños grandes en lo que se refiere a la evaluación. Quizás el EDA no

Cuadro 7.1: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales continuos con aprendizaje UMDA en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 100 individuos.

	$N = 24$	$N = 50$	$N = 74$	$N = 90$
1 ejecución	26,17 - 0023,3	25,95 - 0023,6	25,98 - 0023,9	25,99 - 0024,2
3 ejecuciones	23,81 - 0067,6	23,86 - 0069,4	24,16 - 0070,5	24,02 - 0071,8
5 ejecuciones	23,35 - 0109,8	23,46 - 0113,3	23,36 - 0115,1	23,44 - 0116,0
10 ejecuciones	23,04 - 0212,1	22,88 - 0220,8	22,79 - 0227,0	22,93 - 0227,6
Sin recursión	25,65 - 0045,1	28,60 - 0050,4	28,87 - 0047,3	28,88 - 0046,1

Cuadro 7.2: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales continuos con aprendizaje UMDA en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 500 individuos.

	$N = 120$	$N = 250$	$N = 370$	$N = 450$
1 ejecución	25,55 - 0088,6	25,28 - 0090,8	25,31 - 0093,3	25,42 - 0095,6
3 ejecuciones	23,62 - 0252,7	23,56 - 0260,2	23,76 - 0270,7	23,46 - 0266,9
5 ejecuciones	23,14 - 0404,0	23,08 - 0436,6	22,99 - 0453,2	23,15 - 0443,8
10 ejecuciones	22,84 - 0813,4	22,94 - 0845,1	22,87 - 0865,8	22,87 - 0876,4
Sin recursión	26,79 - 0744,6	28,57 - 0704,8	28,17 - 0673,9	27,96 - 0608,3

es capaz de centrarse en la producción de individuos buenos si la población contiene demasiada variedad. En los Cuadros 7.1 y 7.2 podemos ver los resultados numéricos de estas diez ejecuciones. Los Cuadros 7.3 y 7.4 confirman los resultados para los EDA continuos con aprendizaje MIMIC, y por último los Cuadros 7.5, 7.6, 7.7 y 7.8 lo hacen para los EDA discretos en general.

7.3.5 Función de evaluación del individuo

La función de evaluación es la misma que en el Capítulo 4.

Cuadro 7.3: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales continuos con aprendizaje MIMIC en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 100 individuos.

	$N = 24$	$N = 50$	$N = 74$	$N = 90$
1 ejecución	25,60 - 0022,9	25,74 - 0023,9	25,81 - 0024,1	26,90 - 0025,2
3 ejecuciones	23,51 - 0066,4	23,81 - 0069,3	23,61 - 0069,6	23,84 - 0071,4
5 ejecuciones	23,49 - 0109,7	23,30 - 0113,9	23,59 - 0115,3	23,27 - 0118,1
10 ejecuciones	23,17 - 0213,6	23,02 - 0221,0	22,98 - 0226,3	22,84 - 0229,9
Sin recursión	25,66 - 0046,2	27,78 - 0050,9	28,91 - 0048,6	28,87 - 0046,2

Cuadro 7.4: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales continuos con aprendizaje MIMIC en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 500 individuos.

	$N = 120$	$N = 250$	$N = 370$	$N = 450$
1 ejecución	25,39 - 0089,2	25,31 - 0096,5	25,37 - 0093,9	25,05 - 0093,6
3 ejecuciones	23,39 - 0249,2	23,75 - 0257,4	23,84 - 0269,2	23,62 - 0272,0
5 ejecuciones	23,05 - 0416,5	23,21 - 0453,0	23,15 - 0450,0	22,94 - 0444,1
10 ejecuciones	22,92 - 0802,3	22,97 - 0819,0	22,74 - 0845,4	22,81 - 0855,2
Sin recursión	26,86 - 0746,8	28,24 - 0711,3	27,91 - 0669,5	28,25 - 0607,8

Cuadro 7.5: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales discretos con aprendizaje UMDA en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 100 individuos.

	$N = 24$	$N = 50$	$N = 74$	$N = 90$
1 ejecución	25,22 - 0024,1	26,28 - 0023,9	25,86 - 0023,8	25,37 - 0023,9
3 ejecuciones	23,60 - 0071,4	23,99 - 0071,0	23,87 - 0071,5	23,60 - 0071,4
5 ejecuciones	23,14 - 0119,9	23,02 - 0118,2	23,27 - 0118,5	23,18 - 0120,0
10 ejecuciones	22,88 - 0237,0	23,04 - 0233,9	22,95 - 0236,9	22,88 - 0236,5
Sin recursión	25,19 - 0052,7	25,82 - 0053,8	27,96 - 0051,4	28,60 - 0049,5

Cuadro 7.6: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales discretos con aprendizaje UMDA en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 500 individuos.

	$N = 120$	$N = 250$	$N = 370$	$N = 450$
1 ejecución	24,92 - 0238,4	24,68 - 0157,8	24,94 - 0173,9	25,68 - 0163,5
3 ejecuciones	23,26 - 0826,4	23,24 - 0586,9	23,43 - 0549,5	23,46 - 0517,0
5 ejecuciones	22,78 - 1209,4	22,97 - 0898,8	22,99 - 0824,9	23,14 - 0824,5
10 ejecuciones	22,85 - 2187,8	22,91 - 1833,7	22,91 - 1785,2	23,25 - 1658,6
Sin recursión	23,70 - 0817,8	24,63 - 0795,5	26,68 - 0715,8	27,98 - 0640,7

Cuadro 7.7: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales discretos con aprendizaje MIMIC en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 100 individuos.

	$N = 24$	$N = 50$	$N = 74$	$N = 90$
1 ejecución	26,20 - 0027,8	25,83 - 0026,7	26,26 - 0026,2	26,26 - 0026,5
3 ejecuciones	23,76 - 0081,4	23,91 - 0079,2	24,22 - 0080,4	23,80 - 0079,1
5 ejecuciones	23,35 - 0136,6	23,52 - 0136,1	23,61 - 0135,3	23,38 - 0130,7
10 ejecuciones	22,93 - 0272,6	23,08 - 0271,3	23,00 - 0271,1	23,14 - 0263,5
Sin recursión	25,09 - 0053,1	26,00 - 0053,4	27,67 - 0051,1	28,30 - 0049,1

Cuadro 7.8: Medias de 10 ejecuciones para la red Sparse y EDAs recursivos y tradicionales discretos con aprendizaje MIMIC en el problema de la triangulación (evaluación - tiempo en segundos) para poblaciones de 500 individuos.

	$N = 120$	$N = 250$	$N = 370$	$N = 450$
1 ejecución	25,82 - 0207,3	25,49 - 0215,1	25,43 - 0272,0	26,02 - 0212,5
3 ejecuciones	23,93 - 0647,2	23,62 - 0867,5	24,10 - 0818,3	23,85 - 0839,0
5 ejecuciones	23,38 - 1078,9	23,42 - 1282,6	22,90 - 1391,7	23,52 - 1390,7
10 ejecuciones	22,72 - 2296,8	22,72 - 2656,1	22,76 - 2855,6	23,08 - 2815,2
Sin recursión	23,62 - 0826,1	24,64 - 0795,6	26,80 - 0719,9	27,94 - 0641,5

7.3.6 Planteamiento de los experimentos

En este caso hemos lanzado los experimentos para una población de $M = 100$ y 500 individuos para las dos redes Sparse y Dense. Se han realizado 50 experimentos por población y red, esto es, 50 lanzamientos de cada EDA (UMDA para individuos continuos ($UMDA_c$), UMDA para individuos discretos ($UMDA_d$), MIMIC para individuos continuos ($MIMIC_c$) y MIMIC para individuos discretos ($MIMIC_d$). Cada lanzamiento está parametrizado de la siguiente manera:

- Muestreo - Se realizan 50 muestras por ejecución del EDA (el programa para cuando se alcanza este número). Como se realizan cinco ejecuciones del EDA recursivo (como ya se comentó en la sección dedicada al criterio de parada) tenemos un total de 250 muestras. Otra decisión a tomar es cuándo se realiza la muestra. Si se decide hacerla coincidir con la producción de cada generación, tenemos sólo cincuenta generaciones por ejecución, y esto es muy poco, máxime teniendo en cuenta que cada ejecución se divide en otras muchas debido a la recursión. Se ha decidido imprimir una muestra cada $6 \cdot M$ evaluaciones distintas. En el caso de que el EDA proponga $6 \cdot M$ órdenes seguidos ya evaluados anteriormente, se realiza una muestra para no bloquear la ejecución de una prueba demasiado tiempo. La repetición de órdenes puede llegar a ser grande cuando el tamaño del individuo es pequeño.

- Criterio de parada - El criterio de parada es, como se ha mencionado antes, las 50 muestras por ejecución.

- Grupo de selección - En el Cuadro 7.9 podemos ver el tamaño del grupo de selección utilizado para cada tipo de EDA y basado en los mejores resultados de las pruebas para Sparse. Se ha intentado no usar valores muy extremos: por eso en el caso del $MIMIC_c$ se ha optado por un tamaño de 120 en vez de 450, que es el que da el mejor resultado (ver Cuadro 7.4).

- Nueva población - En cada generación se simula toda la población.

- Otros parámetros - No se utiliza elitismo y la selección de los individuos se realiza por truncación.

  Los experimentos se han realizado en una máquina Sun Fire E2900 con procesadores UltraSPARC IV a 1,2 Ghz y 48 Gb de RAM.

Cuadro 7.9: Tamaños de los grupos de selección para todos los EDAs.

Tamaño de la población	$UMDA_c$	$MIMIC_c$	$UMDA_d$	$MIMIC_d$
100	24	90	50	24
500	370	120	120	370

Cuadro 7.10: Resultados medios de 50 ejecuciones para Sparse y EDAs continuos.

	MIMIC$_c$	UMDA$_c$	Mejor MIMIC$_c$	Mejor UMDA$_c$
M=100	23,28±0,43	23,42±0,35	22,67	22,63
M=500	23,01±0,09	23,20±0,24	22,66	22,66

7.4 Resultados obtenidos

En los Cuadros 7.10 y 7.11 podemos ver los mejores resultados medios, junto con sus varianzas, y las mejores evaluaciones obtenidas para la red Sparse y, respectivamente, los EDA continuos y discretos. En los Cuadros 7.12 y 7.13 podemos ver los mismos resultados para la red Dense. En el Cuadro 7.14 podemos ver qué tipo de EDA (MIMIC o UMDA) ha conseguido los mejores resultados medios, tanto para la red Sparse como para la red Dense y los dos tipos de población. Podemos ver que las diferencias son siempre muy pequeñas. En las Figuras 7.21 y 7.22 vemos una comparación para poblaciones de 500 individuos de todos los tipos de EDA propuestos.

En las Figuras 7.23 y 7.24 podemos comparar el comportamiento para las poblaciones de 100 y 500 individuos para las redes Sparse y Dense. Como se ve, aunque con mayores poblaciones se alcanzan mejores resultados más rápido, tras una serie de muestras la mejora obtenida con poblaciones grandes no es muy acusada, y hay que contar también con la ganancia de tiempo que supone trabajar con poblaciones

Cuadro 7.11: Resultados medios de 50 ejecuciones para Sparse y EDAs discretos.

	MIMIC$_d$	UMDA$_d$	Mejor MIMIC$_d$	Mejor UMDA$_d$
M=100	23,28±0,17	23,23±0,20	22,63	22,66
M=500	23,30±0,25	23,15±0,22	22,61	22,64

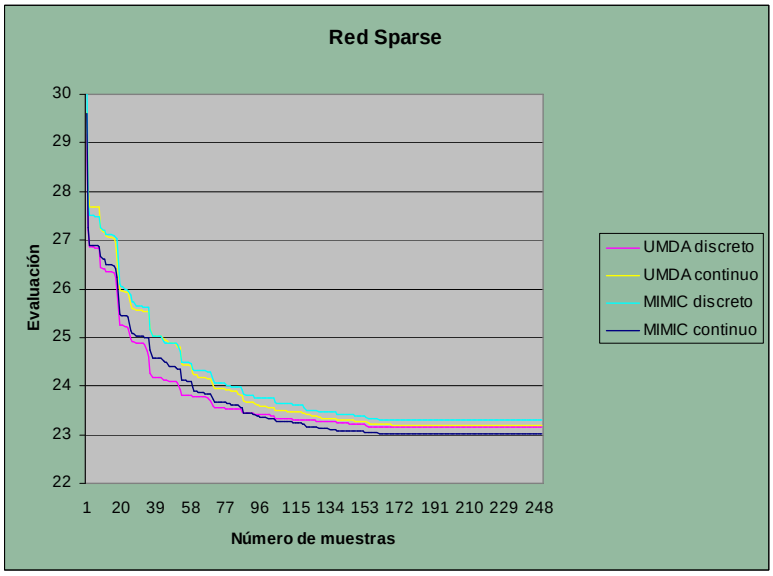

Figura 7.21: Comparación para Sparse de los EDA recursivos continuos y discretos para poblaciones de 500 individuos.

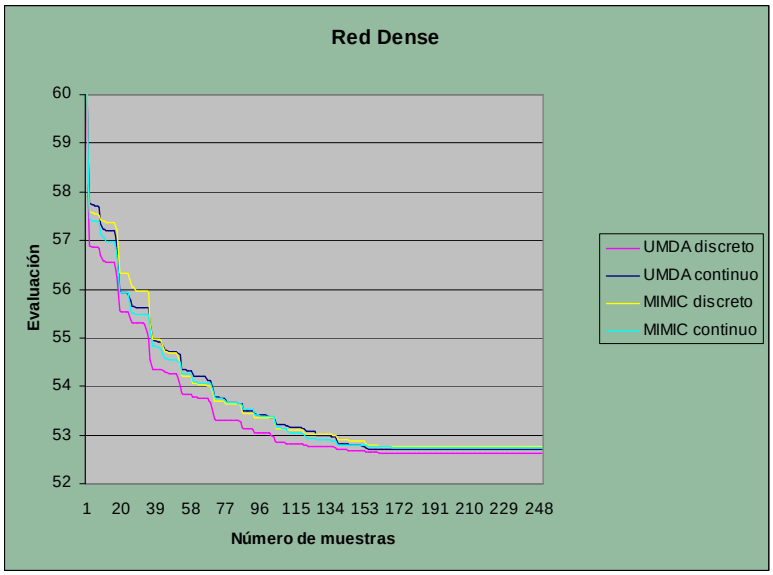

Figura 7.22: Comparación para Dense de los EDA recursivos continuos y discretos para poblaciones de 500 individuos.

Cuadro 7.12: Resultados medios de 50 ejecuciones para Dense y EDAs continuos.

	MIMIC$_c$	UMDA$_c$	Mejor MIMIC$_c$	Mejor UMDA$_c$
M=100	53,09±1,13	53,04±1,05	51,15	51,15
M=500	52,73±1,58	52,70±1,44	50,88	50,88

Cuadro 7.13: Resultados medios de 50 ejecuciones para Dense y EDAs discretos.

	MIMIC$_d$	UMDA$_d$	Mejor MIMIC$_d$	Mejor UMDA$_d$
M=100	53,04±0,85	52,88±1,19	50,88	50,99
M=500	52,75±0,73	52,62±1,31	50,89	50,88

pequeñas (compárense, en los Cuadros 7.1 y 7.2, los $23,6$ segundos de media para 100 individuos y una ejecución recursiva con $N = 50$ con los $88,6$ segundos para 500 individuos en las mismas condiciones y con $N = 120$), con una evaluación media similar. En las Figuras 7.25 y 7.26 podemos ver comparaciones equivalentes para los EDA discretos.

7.5 Conclusiones

Podemos extraer las siguientes conclusiones de los resultados de los experimentos mostrados en la sección anterior:

Cuadro 7.14: Mejor EDA para cada combinación de tamaño de población y tipo de variable.

	Continuo Sparse	Discreto Sparse	Continuo Dense	Discreto Dense
M=100	MIMIC	UMDA	UMDA	UMDA
	$(23,28 < 23,42)$	$(23,23 < 23,28)$	$(53,04 < 53,09)$	$(52,88 < 53,04)$
M=500	MIMIC	UMDA	UMDA	UMDA
	$(23,01 < 23,20)$	$(23,15 < 23,30)$	$(52,70 < 52,73)$	$(52,62 < 52,75)$

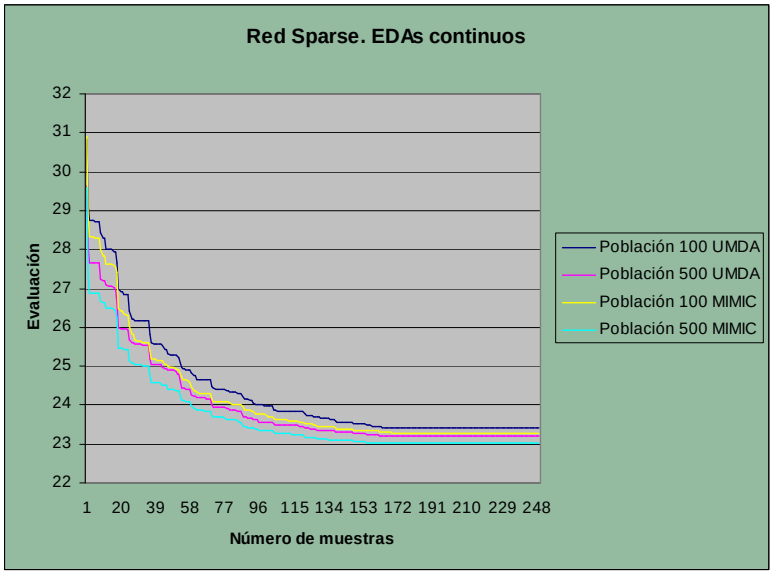

Figura 7.23: Evolución de la evaluación obtenida con el EDA continuo a lo largo de las 250 muestras para la red Sparse y poblaciones de 100 y 500 individuos.

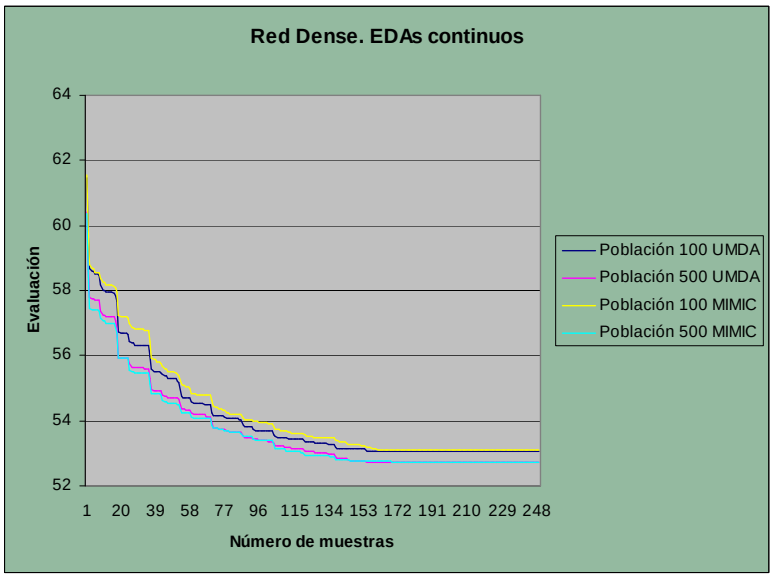

Figura 7.24: Evolución de la evaluación obtenida con el EDA continuo a lo largo de las 250 muestras para la red Dense y poblaciones de 100 y 500 individuos.

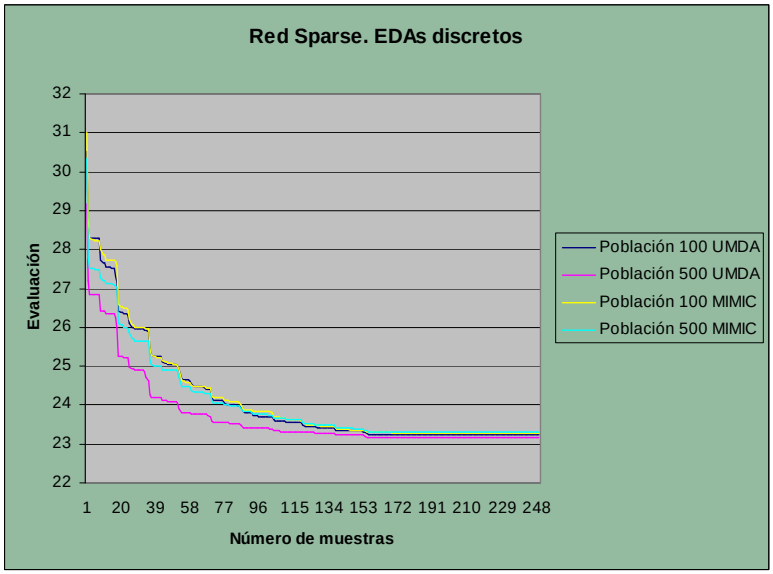

Figura 7.25: Evolución del EDA discreto a lo largo de las 250 muestras para la red Sparse y poblaciones de 100 y 500 individuos.

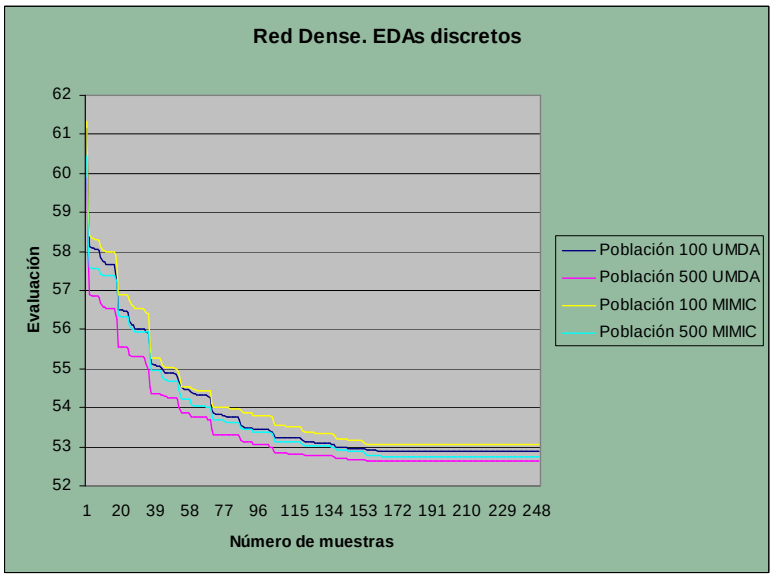

Figura 7.26: Evolución del EDA discreto a lo largo de las 250 muestras para la red Dense y poblaciones de 100 y 500 individuos.

- Por un lado, parece clara la superioridad de los EDAs recursivos con respecto a los tradicionales en cuanto a las medias de las evaluaciones, y también en cuanto al mejor individuo encontrado, como se puede ver por ejemplo en los Cuadros 7.15 y 7.16 para las redes Sparse y Dense respectivamente. Tan solo el UMDA$_d$ tradicional obtiene resultados medios similares que el recursivo, y tan solo en poblaciones de 500 individuos. En el test de suma de rangos de wilcoxon para comparar medianas, sin embargo, el UMDA$_d$ recursivo es superior al tradicional en la red Sparse con un $p-$valor de $0,0003$, mientras que con la red Dense no hay diferencias estadísticamente significativas ($p-$valor de $0,88$). Las dispersiones son similares. Especialmente significativa es la mejora para la población de 100 individuos. Por lo tanto, la estrategia de *dividir* aleatoriamente el individuo en segmentos y combinar la evolución sobre esos segmentos con la evolución del individuo completo, ¿podría ser siempre mejor? No es exactamente así. Como podemos ver en los valores de la evaluación en los Cuadros 7.13, 7.14, 7.15, 7.16, 7.17, 7.18, 7.19 y 7.20, el EDA tradicional es muy sensible al tamaño del grupo de selección: elegirlo mal puede provocar resultados malos. En cambio, en los EDA recursivos su influencia es mínima, y en esto sí son claramente superiores. Quizás en otro tipo de problemas la diferencia de estabilidad entre el EDA tradicional y el recursivo sea menor. Por ejemplo, en el problema del aprendizaje estructural hay un algoritmo asociado, el K2, que puede minimizar la distancia entre la versión tradicional y la recursiva.

- Los EDA recursivos tienen un comportamiento razonablemente similar independientemente del tamaño de la población, y aunque haya diferencias estadísticamente significativas, éstas no son muy grandes (en el caso de la red Sparse, por ejemplo, tenemos un $p-$valor en la comparación de las medianas de los EDAs para poblaciones de 100 y 500 individuos de $0,01$ para MIMIC$_c$, $0,96$ para MIMIC$_d$, $0,06$ para UMDA$_c$ y $0,11$ para UMDA$_d$, luego sólo tenemos un único $p-$valor por debajo de $0,05$). Como hemos comentado antes, el comportamiento también parece ser similar independientemente del tamaño del grupo de selección utilizado en el EDA, e incluso parece también no depender demasiado del tipo de EDA (UMDA o MIMIC) y de si las variables son contínuas o discretas. En concreto, y sobre esta última afirmación, tenemos que para la red Sparse y poblaciones de 500 individuos, solo se encuentran diferencias significativas con un p-valor inferior a $0,05$ en el test de suma de rangos de Wilcoxon para comparar medianas entre MIMIC$_c$ y MIMIC$_d$ $(0,0008)$ y entre MIMIC$_c$ y UMDA$_d$ $(0,03)$, no habiendo diferencias significativas en ninguna otra com-

paración. En el caso de la red Dense, esta tendencia queda más clara, ya que no hay diferencias significativas entre ninguno de los algoritmos (el p-valor más pequeño, entre UMDA$_c$ y MIMIC$_d$, resulta ser $0,53$). Las pequeñas diferencias entre los distintos algoritmos pueden apreciarse gráficamente en las Figuras 7.27 y 7.28. Estos últimos resultados repercuten en el tiempo de ejecución, ya que la combinación de poblaciones pequeñas, EDAs continuos y UMDA es mucho menos costosa, computacionalmente hablando, que cualquier otra. Los EDAs discretos van a necesitar además poblaciones muy grandes puesto que el tamaño del individuo es también muy grande, y por si fuera poco alguno de los genes llega a tener valores muy altos (los asociados a números primos grandes), situación que lastra el uso de la red Bayesiana asociada al EDA.

- Si se necesita un valor muy bueno, siempre se pueden realizar varias ejecuciones del EDA recursivo sobre la misma cache para refinar constantemente el resultado (en las pruebas hemos obtenido mejoras, a costa del tiempo de ejecución eso si, con hasta 10 ejecuciones).

- Se ha podido constatar que hay mejoras significativas en el valor del mejor individuo cuando se está evolucionando sobre individuos cortos, lo cual hace que la cache se enriquezca en estas fases con individuos buenos, esto es, no hay unos pocos individuos buenos y demasiados malos, cosa que puede hacer que la red Bayesiana no se ajuste al subespacio de búsqueda deseado todo lo que quisiéramos.

- De las tres fases del EDA más costosas computacionalmente hablando, esto es, el aprendizaje de la distribución de probabilidad, la simulación de la red Bayesiana y la evaluación de los individuos, solo las dos primeras son mejoradas por los EDAs recursivos, ya que se ven implicados en la red individuos de menor tamaño. En cambio, a la hora de la evaluación del individuo, hay que tener en cuenta que el algoritmo $Evaluar_\beta$ siempre acaba haciendo una llamada a $Evaluar_\alpha$, por lo que no hay mejora de velocidad alguna en este caso. De todas maneras, suele ser fácil paralelizar la evaluación de los individuos descomponiendo la función en grupos de variables, y además es la fase de aprendizaje la que más tiempo consume.

Cuadro 7.15: Comparación de los resultados medios de 50 ejecuciones de los EDA recursivos y los tradicionales para Sparse.

	Recursivo	Tradicional	Mejor recursivo	Mejor tradicional
M=100 y UMDA$_c$	23,42±0,35	26,80±2,34	22,63	23,48
M=500 y UMDA$_c$	23,20±0,24	27,43±0,33	22,66	26,26
M=100 y UMDA$_d$	23,23±0,20	26,55±0,90	22,66	24,38
M=500 y UMDA$_d$	23,15±0,22	23,35±0,09	22,64	22,66
M=100 y MIMIC$_c$	23,28±0,43	26,79±2,35	22,67	23,48
M=500 y MIMIC$_c$	23,01±0,09	27,55±0,23	22,66	26,65
M=100 y MIMIC$_d$	23,28±0,17	27,61±1,61	22,63	24,20
M=500 y MIMIC$_d$	23,30±0,25	27,72±0,63	22,61	25,73

Cuadro 7.16: Comparación de los resultados medios de 50 ejecuciones de los EDA recursivos y los tradicionales para Dense.

	Recursivo	Tradicional	Mejor recursivo	Mejor tradicional
M=100 y UMDA$_c$	53,04±1,05	58,01±1,23	51,15	55,26
M=500 y UMDA$_c$	52,70±1,44	57,97±0,31	50,88	56,51
M=100 y UMDA$_d$	52,88±1,19	53,54±1,47	50,99	51,23
M=500 y UMDA$_d$	52,62±1,31	52,58±1,15	50,88	51,11
M=100 y MIMIC$_c$	53,09±1,13	57,87±0,95	51,15	55,63
M=500 y MIMIC$_c$	52,73±1,58	57,92±0,49	50,88	56,19
M=100 y MIMIC$_d$	53,04±0,85	54,50±0,92	50,88	52,00
M=500 y MIMIC$_d$	52,75±0,73	54,26±1,02	50,89	52,32

7.6 Propuesta de paralelización

Dejando a un lado las paralelizaciones, por decirlo de alguna manera, *inmediatas* de los algoritmos evolutivos (38), vemos que en todos los intentos de paralelización antes comentados que se aplican a fases concretas del EDA hay una descomposición del dominio combinada con una descomposición funcional inicial (62). Cada diseño paralelo se aplica a una determinada fase del EDA de manera independiente, pero no de manera global a todo el EDA. Esto hace que el flujo paralelo se interrumpa

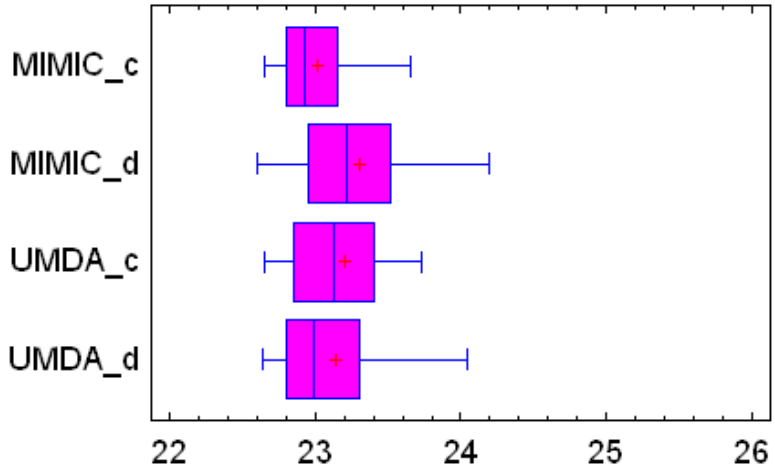

Figura 7.27: Gráfico de cajas y bigotes para comparar todos los EDAs aplicados a la red Sparse para poblaciones de 500 individuos. Los datos están divididos en cuatro áreas de frecuencia iguales (cuartiles). La caja engloba el 50 % central y la mediana aparece dibujada como una línea dentro de la caja.

forzosamente en muchas ocasiones, esto es, las fases paralelas no son todo lo *largas* que sería deseable, y transfieren el flujo global del programa al proceso principal (que no es paralelo y que es el que crea los demás) varias veces en cada generación, que era precisamente uno de los problemas de la estrategia de los evolutivos paralelos de tipo maestro-esclavo. Lo ideal sería un sistema que paralelizara el EDA completamente, de manera global, en el que el flujo de ejecución se mantuviera paralelo desde el principio hasta el final, y en el que el ratio de intercambio de mensajes entre procesos fuera de alguna manera seleccionable, algo parecido al ratio de migración de los evolutivos paralelos de grado grueso. Sin embargo, la opción de los de grano grueso, como tal, tampoco parece muy deseable, ya que la fase de aprendizaje sigue siendo extremadamente costosa porque no hay una simplificación del contexto. Esta falta de simplificación puede hacer que no podamos aplicar el algoritmo en determinados contextos. Reducir el tamaño de población para que el aprendizaje sea más rápido puede ser, como hemos comentado antes, peligroso.

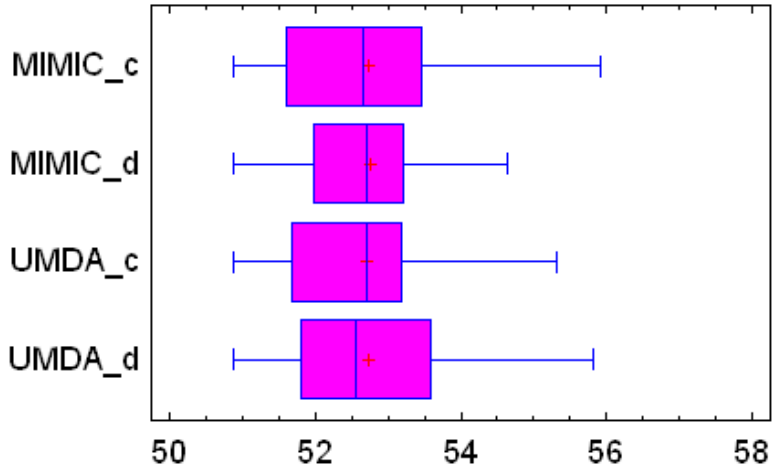

Figura 7.28: Gráfico de cajas y bigotes para comparar todos los EDAs aplicados a la red Dense para poblaciones de 500 individuos. Ver Figura 7.27 para una descripción más detallada del gráfico.

En (115), los distintos procesos forman diferentes topologías en función de la ausencia o no de caminos para transmitir información entre ellos (anillo y estrella). La información que se migra se limita a determinados individuos o a *modelos*, esto es, distribuciones de probabilidad. En el caso de los modelos, la manera en la que se combinan estos puede variar. Mediante esta táctica, son varios los procesos evolucionando sobre el mismo problema y *enriqueciendo* a los demás con sus *hallazgos*. Aunque esto no acelera en principio el funcionamiento de ninguno de los procesos como tales, si que puede acelerar la convergencia global o la velocidad de obtención de un resultado bueno. Pero esto no soluciona el problema de los contextos muy complejos en los que cada proceso es ya muy costoso computacionalmente hablando.

Sin embargo, podemos intentar combinar la versatilidad en cuanto a ratio de migración de los evolutivos de grado grueso, que nos ayuda a controlar el intercambio de información entre procesos, con la simplificación del contexto de los EDA recursivos, que nos ayuda a controlar el coste de la etapa de aprendizaje. Además, como hemos visto en los experimentos, el comportamiento de los EDA recursivos es relativamente

independiente de varios parámetros, incluído (hasta cierto punto, claro) el tamaño de población, aspecto que mediatizaba demasiado el diseño de los evolutivos paralelos en general. Con respecto a otros intentos de paralelización no inmediata, ya no tenemos que preocuparnos aquí por la paralelización de una función de evaluación de la red, es decir, el método es más general y se aplica al algoritmo EDA completo.

7.6.1 El algoritmo paralelo

Parece evidente que cada ejecución recursiva podría ser llevada a cabo en un procesador independiente. Cada vez que el conjunto de nodos se divide en dos, cada subconjunto sería desarrollado de manera aislada. No podemos realizar la primera evolución sobre el subconjunto de nodos completo actual antes de dividirlo y llevar a cabo las dos llamadas recursivas, como aparece en el pseudocódigo de la figura 7.9, sino las tres cosas simultáneamente en tres procesadores distintos. Esta es la situación ideal. El esquema de paralelización aparece en la figura 7.29. El grado de la topología es como máximo de tres para los nodos internos del árbol binario, es decir, la comunicación para un proceso concreto se limita a:

- Recibir del proceso padre m_r y G_r.
- Enviar a los dos procesos hijos m_{r+1}, G_{r+1} y G'_{r+1}

Aunque como veremos más adelante, será necesario intercambiar otro tipo de información, básicamente sobre terminación de procesos.

En la Figura 7.30 podemos ver una primera transformación a su versión paralela del algoritmo $EdaRecursivo(m_r, G_r)$. Como el número de recursiones es fácil de calcular, podríamos realizar a priori la fase de mapeo (asociación de los distintos procesos a los procesadores disponibles). Como se ve, hemos añadido un argumento más a *EdaBasico*, que es el criterio de parada a tener en cuenta y que explicaremos más detenidamente en la siguiente sección, así como la interrogación que aparece en este argumento en la segunda llamada a *EdaBasico*.

El algoritmo, tal y como aparece en la Figura 7.30, plantea dos problemas principales:

- Por un lado, no todas las ejecuciones recursivas tardan lo mismo. En la primera, por ejemplo, se evoluciona sobre el individuo completo, mientras que en la última se puede estar evolucionando sobre sólo 4 genes.

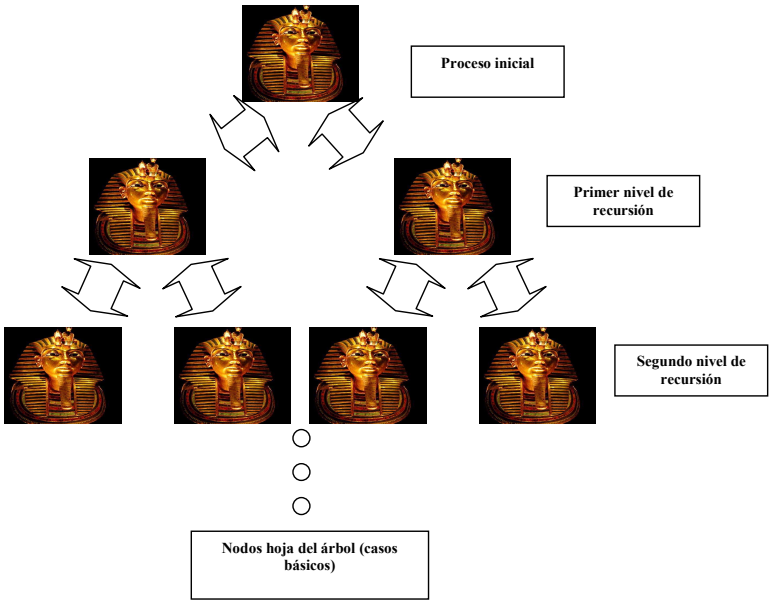

Figura 7.29: Esquema de paralelización de tipo divide y vencerás que podría aplicarse a los EDA recursivos.

- Una de las principales mejoras de los EDA Recursivos consistía en la peculiar carga de la cache al comienzo del EDA Básico, en la que se intentaba aprovechar al máximo la información sobre los órdenes buenos que ya teníamos de ejecuciones anteriores. Sin embargo, ¿Qué ocurre ahora con las caches? ¿Hay una sola o tantas como procesos? En el segundo caso, ¿Cómo podemos aprovechar la información de cada cache en todas las ejecuciones del EDA Básico?

7.6.2 Posible solución al problema de la variabilidad del tiempo de ejecución

Ya que hemos de esperar a que termine el proceso mas largo en el tiempo, el primer problema podría ser resuelto haciendo que cada invocación del EDA recursivo siguiera produciendo generaciones mientras la invocación más larga no haya terminado. Esto reduce el tiempo de ejecución del EDA al asociado a las generaciones que queramos que transcurran para el subconjunto de mayor número de nodos (esto es, para el de tamaño m), que pueden ser tan pocas como queramos. Es decir, cada invocación de *EdaParalelo* sigue produciendo generaciones hasta que vea que su proceso padre ha

```
Procedimiento EdaParalelo ( m_r, G_r )
  p ← MiProcesoPadre
  Si m_r < 4 entonces
    no hacer nada
  Sino
    Seleccionar trunc(m_r/2) nodos de G_r aleatoriamente.
  Sea G_{r+1} ⊂ G_r el conjunto de los trunc(m_r/2) nodos seleccionados
    m_{r+1} ← trunc(m_r/2)
    h_1 ← Crear proceso hijo para EdaParalelo ( m_r − m_{r+1}, G_{r+1} )
    G'_{r+1} ← G \ G_{r+1}
    Crear proceso hijo para EdaParalelo ( m_{r+1}, G'_{r+1} )
    EdaBasico ( m_r, G_r, FinEdaBasicoPadre(p) )
    Enviar FinEdaBasicoPadre a los dos procesos hijos h_1 y h_2
    Esperar FinHijos(h_1, h_2)
  FinSi
  EdaBasico ( m_r, G_r, ? )
  Enviar FinHijo al proceso padre
```

Figura 7.30: Primera versión del pseudocódigo del EDA paralelo sin tener en cuenta el número de procesadores.

terminado su llamada a *EdaBasico*. Esta podría ser la nueva condición de parada de *EDABasico*. Esta es la opción que aparece en el algoritmo de la figura 7.30. Sin embargo, el primer proceso, aquel que trabaja con el conjunto de m nodos, tendrá una condición de parada especial: como no tiene padre, detiene su ejecución tras producir un número g de generaciones prefijado. En una máquina con muchos procesadores de tipo cluster, esta sería la solución más adecuada.

Sin embargo, si tenemos pocos procesadores, podemos ejecutar las últimas evoluciones (que son de subconjuntos de nodos que tienen ya un tamaño pequeño) en un mismo procesador de una manera muy sencilla: llamando en un momento dado al procedimiento *EdaRecursivo* en lugar de a *EdaParalelo*. En un equipo de memoria compartida, donde el número de procesadores es típicamente menor, esta sería la mejor opción. Esta nueva versión del algoritmo paralelo quedaría tal y como aparece en la figura 7.31. Sin embargo, esta versión aún tiene problemas graves, como veremos más adelante.

```
Procedimiento EdaParalelo ( m_r, G_r )
  p ← MiProcesoPadre
  Si m_r < 4 entonces
    no hacer nada
  Sino
    Si no hay dos procesadores libres entonces
      EdaBasico ( m_r, G_r )
      Seleccionar trunc(m_r/2) nodos de G_r aleatoriamente.
      Sea G_{r+1} ⊂ G_r el conjunto de los trunc(m_r/2) nodos seleccionados
      m_{r+1} ← trunc(m_r/2)
      EdaRecursivo ( m_{r+1}, G_{r+1} )
      G'_{r+1} ← G \ G_{r+1}
      EdaRecursivo ( m_r − m_{r+1}, G'_{r+1} )
    Sino
      Seleccionar trunc(m_r/2) nodos de G_r aleatoriamente.
      Sea G_{r+1} ⊂ G_r el conjunto de los trunc(m_r/2) nodos seleccionados
      m_{r+1} ← trunc(m_r/2)
      h_1 ← Crear proceso hijo para EdaParalelo ( m_{r+1}, G_{r+1} )
      G'_{r+1} ← G \ G_{r+1}
      h_2 ← Crear proceso hijo para EdaParalelo ( m_r − m_{r+1}, G'_{r+1} )
      EdaBasico ( m_r, G_r, FinEdaBasicoPadre(p) )
    Enviar FinEdaBasicoPadre a los dos procesos hijos h_1 y h_2
    Esperar FinHijos(h_1, h_2)
    FinSi
  FinSi
  EdaBasico ( m_r, G_r, ? )
  Enviar FinHijo al proceso padre
```

Figura 7.31: Pseudocódigo del EDA paralelo teniendo en cuenta el número de procesadores disponible.

7.6.3 Posible solución al problema de la independencia de las caches

En el algoritmo *EdaRecursivo*, tras la primera llamada a *EdaBasico*, hay una grabación de la cache cuyo contenido es luego incorporado en las llamadas al proce-

dimiento *EdaBasico* de las dos invocaciones recursivas. Sin embargo, en el caso de la versión paralela, no podemos esperar a que termine la primera llamada a *EdaBasico* antes de crear los dos procesos hijo, porque entonces la paralelización sería bastante mala. En un primer momento podemos pensar que los procesos hijo harán uso de una cache que no procede de la ejecución del *EdaBasico* del proceso padre, sino de la del proceso abuelo. Pero en realidad no es así, ya que la distribución de procesos en los distintos procesadores se hace nada más arrancar el algoritmo, es decir, todas las primeras invocaciones de *EdaBasico* son simultáneas y parten de una cache vacía. ¿Podemos permitirnos esto? ¿Se perderá gran parte de la eficacia del algoritmo debido a esta primera llamada a *EdaBasico* sobre una cache vacía, esto es, con una primera población siempre aleatoria?

En cualquier caso, tras la finalización de los procesos hijo, el proceso padre puede tener disponibles sus caches para mezclar la suya con éstas y aprovechar toda esta información en la segunda llamada al procedimiento *EdaBasico*. Esta segunda llamada, que parecía poco efectiva en el caso del EDA Recursivo en relación a su coste computacional (sobre todo en las primeras invocaciones recursivas, cuando el tamaño del individuo es grande) se revela, en la versión paralela, vital para incorporar toda la información producida por los distintos procesos. Sin embargo, esta espera a la finalización de los dos procesos hijo antes de lanzar una nueva invocación de *EdaBasico* es un problema grave.

Supongamos que el proceso h ha terminado su primera llamada a *EdaBasico* porque ha recibido de su proceso padre p el mensaje de fin de *EdaBasico*. En este momento es cuando envia a su vez el mensaje de finalización de *EdaBasico* a sus dos procesos hijo n_1 y n_2. Una vez acaban éstos, el proceso h realiza una nueva llamada a *EdaBasico*, pero hay que tener en cuenta que su proceso padre p se ha quedado esperando su finalización como hijo: durante toda esta segunda llamada a *EdaBasico* tenemos a un proceso parado sin hacer nada: p. Además, el criterio de parada que hemos usado hasta ahora no sirve para esta segunda llamada a *EdaBasico*: de ahí la interrogación que aparece en el argumento en el que se explicita el criterio de parada en la figura 7.31. Podemos solucionar esto tomando las siguientes decisiones:

- Cuando el proceso h recibe el mensaje de fin de *EdaBasico* del proceso p, no detiene la ejecución de su propio *EdaBasico*, sino que se limita a enviar el mensaje de fin de *EdaBasico* a sus procesos hijos y sigue produciendo generaciones hasta que los procesos hijos finalizan su ejecución.

- La señal que desencadena los sucesivos envíos de la señal de fin de *EdaBasico* es la finalización del *EdaBasico* del primer proceso, aquel que utiliza los sub-

conjuntos de nodos de tamaño m y cuyo criterio de parada consiste en la producción de un número fijo de generaciones. Para que este primer proceso tampoco quede esperando la finalización de los hijos, puede enviar la señal de fin de *EdaBasico* una vez transcurridas la mitad de las generaciones, y quedarse produciendo generaciones (en su primera llamada a *EdaBasico*) hasta la finalización de los hijos.

- La segunda llamada al *EdaBasico* de h no puede tener como criterio de parada la finalización del *EdaBasico* de p, que de hecho está esperando el fin de h para hacer a su vez la segunda llamada. Aquí lo que podemos hacer es almacenar el número de generaciones producidas por la primera llamada, y emplearlo como criterio de parada de la segunda, cosa que está en concordancia con la decisión tomada en el párrafo anterior.

- Los procesos hoja del árbol no tienen procesos hijos, por lo que en cuanto reciben la señal de fin de *EdaBasico* de sus padres, terminan directamente. Así, estos padres pueden iniciar la ejecución de sus segundas llamadas a *EdaBasico*.

- Téngase en cuenta que la segunda llamada a *EdaBasico* del primer proceso se realiza *en solitario*, es decir, que cuando comienza todos los demás procesos han terminado.

- Ahora, los distintos procesos han de enviarse el contenido de la cache, que no es más que los M individuos mejores que el EDA ha encontrado, puesto que la ejecución de cada uno de ellos es completamente independiente. Esto podría ser un problema si M es grande y la máquina donde ejecutamos el EDA paralelo es de memoria distribuida, ya que fuerza un intercambio de información considerable entre distintos nodos a través de la red de datos, de respuesta mucho más lenta que la memoria RAM. En un equipo con memoria compartida, todas las caches podrían residir en memoria y lo único que se transmiten los distintos procesos son punteros. Por fortuna, este envío de la cache tiene lugar en muy pocas ocasiones, tan solo al final de cada proceso.

Todas estas decisiones quedan ya incorporadas en las versiones ya definitivas de los procedimientos *EdaBasicoParalelo* [7] (figura 7.33), *EdaRecursivo* (figura 7.32) y el pseudocódigo principal del EDA (figura 7.34).

[7] Hemos cambiado el nombre del procedimiento para que no se confunda ahora con el *EdaBasico* de la versión recursiva.

Procedimiento **EdaParalelo** (m_r, G_r, ζ_r^α)
  Si $m_r < 4$ *entonces*
    no hacer nada
  Sino
    Si no hay dos procesadores libres *entonces*
      $R \leftarrow Cierto$
      EdaBasico (m_r, G_r, ζ_r^α)
      Seleccionar $trunc(\frac{m_r}{2})$ nodos de G_r aleatoriamente
      Sea $G_{r+1} \subset G_r$ el conjunto de los $trunc(\frac{m_r}{2})$ nodos seleccionados
      $m_{r+1} \leftarrow trunc(\frac{m_r}{2})$
      EdaRecursivo (m_{r+1}, G_{r+1}, ζ_r^α)
      $G'_{r+1} \leftarrow G \setminus G_{r+1}$
      EdaRecursivo ($m_r - m_{r+1}$, G'_{r+1}, ζ_r^α)
    Sino
      Seleccionar $trunc(\frac{m_r}{2})$ nodos de G_r aleatoriamente
      Sea $G_{r+1} \subset G_r$ el conjunto de los $trunc(\frac{m_r}{2})$ nodos seleccionados
      $m_{r+1} \leftarrow trunc(\frac{m_r}{2})$
      $\zeta_{r+1}^\alpha \leftarrow \varnothing$
      $h_1 \leftarrow$ Crear proceso hijo para EdaParalelo (m_{r+1}, G_{r+1}, ζ_{r+1}^α)
      $G'_{r+1} \leftarrow G \setminus G_{r+1}$
      $\zeta_{r+1}^{\alpha'} \leftarrow \varnothing$
      $h_2 \leftarrow$ Crear proceso hijo para EdaParalelo (m_{r+1}, G'_{r+1}, $\zeta_{r+1}^{\alpha'}$)
      EdaBasicoParalelo (m_r, G_r, $FinHijos(h_1, h_2)$, g, ζ_r^α, h_1, h_2)
      Dejar en ζ_r^α los M mejores individuos de ζ_r^α, ζ_{r+1}^α y $\zeta_{r+1}^{\alpha'}$
    FinSi
  FinSi
  Si R es cierto *entonces* EdaBasico (m_r, G_r, ζ_r^α)
  Sino EdaBasicoParalelo (m_r, G_r, $FinGeneraciones(g)$, g, ζ_r^α)
  FinSi
  Enviar $FinHijo$ al proceso padre

Figura 7.32: Pseudocódigo definitivo del EDA paralelo. Téngase en cuenta que g contendrá el número de generaciones transcurridas en la primera llamada recursiva a $EdaBasicoParalelo$, y que ζ_r^α está vacía antes de esta primera llamada. Vemos que en la segunda llamada al $EdaBasicoParalelo$ no hace falta enviar los hijos como argumento.

Procedimiento **EDABasicoParalelo** (m_r, G_r, C, g, $\zeta^{\alpha}_{r_{viejo}}$, h_1, h_2)
Si $\zeta^{\alpha}_{r_{viejo}} \neq \varnothing$ *entonces* InicializarPoblacionUsandoCache ($\zeta^{\alpha}_{r_{viejo}}$, D_0, G_r, i^{α}_{mejor})
Sino
$D_0 \leftarrow$ Generar M individuos aleatoriamente (la población inicial)
$\zeta^{\alpha}_r \leftarrow \varnothing$
FinSi
$l \leftarrow 1$
$g \leftarrow 0$
Si El proceso actual no tiene hijos *entonces*
$C \leftarrow FinEdaBasicoPadre$
FinSi
Repetir
Si El proceso actual no tiene padre *entonces*
Si Hemos alcanzado la mitad de las generaciones que
aparecen en C y h_1 y h_2 existen *entonces*
Enviar $FinEdaBasicoPadre$ a los dos procesos hijo h_1 y h_2
FinSi
Sino
Si h_1 y h_2 existen y el proceso padre ha enviado $FinEdaBasicoPadre$ *entonces*
Enviar $FinEdaBasicoPadre$ a los procesos hijo h_1 y h_2
FinSi
FinSi
Para cada individuo $i^{\beta}_{ind} \in D_{l-1}$ *hacer*
$Evaluar^{\beta}(i^{\alpha}_{mejor}, i^{\beta}, i^{\gamma}, s^{\beta})$
Si s^{β} es la mejor evaluación hasta el momento *entonces*
insertar i^{γ} en ζ^{α}_r en el primer puesto
Si $\|\zeta^{\alpha}_r\| > M$, quitar el individuo peor de ζ^{α}_r (el último)
FinSi
FinPara
$D^{s}_{l-1} \leftarrow$ Seleccionar $N \leq M$ individuos de D_{l-1} utilizando un
método de selección prefijado
$\rho_l(\boldsymbol{z}) \leftarrow$ Estimar la distribución de probabilidad de $\boldsymbol{z}$ dado D^{s}_{l-1}
$D_l \leftarrow$ Simular M individuos (la nueva población) a partir de $\rho_l(\boldsymbol{z})$
$l \leftarrow l + 1$; $g \leftarrow g + 1$
Hasta que C sea satisfecho
$\zeta^{\alpha}_{r_{viejo}} \leftarrow \zeta^{\alpha}_r$

Figura 7.33: Pseudocódigo del bucle básico EDA para la versión paralela definitiva.

EDA
$\zeta^{\alpha} \leftarrow \varnothing$
$EdaParalelo(m, G, \zeta^{\alpha})$
Devolver c_0^{α}

Figura 7.34: Pseudocódigo principal del EDA paralelo.

Capítulo 8

Conclusiones y Trabajos Futuros

'En el círculo se confunden el principio y el fin.'

Heráclito de Efeso

En este trabajo se han expuesto los resultados de la aplicación de los algoritmos de estimación de distribuciones en problemas combinatorios con espacios de búsqueda grandes, así como un análisis de éstos y su comparación con el comportamiento de otros tipos de paradigmas de computación. Todos los problemas seleccionados han estado relacionados con las redes Bayesianas. En el capítulo 4 hemos aplicado los EDA al problema de la triangulación, fase clave para realizar la propagación de la evidencia, buscando en un espacio de órdenes. En el capítulo 5 hemos intentado realizar el aprendizaje estructural de la red a partir de una base de datos de casos con ayuda del algoritmo K2 de Lauritzen y Spiegelhalter, buscando también en un espacio de órdenes. En el capítulo 6 hemos tratado de resolver el problema del MAP, aquí buscando directamente en el espacio de explicaciones más probables, y no en espacios de órdenes. Por último, en el capítulo 7, hemos propuesto una manera general de solventar los problemas de eficiencia y calidad de respuesta de los EDA cuando se enfrentan a algunas dificultades, propuesta que, además, es paralelizable.

8.1 Conclusiones

Podemos extraer las siguientes conclusiones de los resultados obtenidos a lo largo de todos los experimentos de esta tesis:

- Los resultados de los EDA son válidos y superan siempre a los experimentos aleatorios en todos los problemas afrontados.

- La parametrización del EDA es una de las dificultades que más mediatiza los resultados, como hemos podido ver en el capítulo 4. Una mala decisión, por ejemplo en el tamaño del grupo de selección a partir del cual se genera la red, puede hacer que para determinados problemas los resultados sean no todo lo buenos que podrían ser.

- En algunos problemas, como por ejemplo el aprendizaje de redes a partir de bases de datos de casos, la unión de los EDA con otros métodos de búsqueda que son ya de por si bastante válidos (el algoritmo K2 en este caso) arrojan resultados muy buenos y comparables con cualquier otro método heurístico. Sin embargo, en otros problemas en los que los EDA han de enfrentarse directamente a un espacio de búsqueda grande, como el problema del MAP en el capítulo 6, tienden a caer en máximos/mínimos locales o no acaban de dirigirse claramente hacia el subespacio en el que se encuentran los mejores individuos. Si bien siguen dando unos resultados razonables, no pueden competir con otras estrategias evolutivas como son los algoritmos genéticos (aunque en este último caso, si son capaces de obtener un mayor número de individuos buenos, cosa que en el problema concreto de la abducción parcial es casi tan interesante como obtener la mejor explicación).

- La estrategia de la división del individuo nos proporciona, tal y como hemos visto en el capítulo 7, dos ventajas principales, por lo menos en la aplicación a la triangulación, como son la relativa independencia de la parametrización a la hora de conseguir resultados buenos y el aumento claro de la eficiencia del algoritmo.

- Esta última estrategia nos proporciona una nueva forma de paralelización de los algoritmos EDA que no depende del problema al que se aplican y que no es la usual (no esta asociada a la función de evaluación, o a la fase de evaluación de los individuos en general, sino al algoritmo EDA completo).

8.2 Trabajos futuros

Las líneas de investigación futuras que se derivan del presente trabajo serían principalmente dos:

- Por un lado, habría que tratar de confirmar, para otros tipos de problemas combinatorios, los buenos resultados que los EDA recursivos han conseguido para

el caso particular de la triangulación. En concreto, sería necesario confirmar el aumento de la eficiencia y la relativa independencia de la parametrización, que son las dos principales aportaciones de esta estrategia.

- Por otro lado, la implementación y análisis del comportamiento del algoritmo paralelo propuesto en el capítulo 7 también sería muy interesante, ya que posibilitaria la aplicación de los EDA a problemas reales, que implican espacios de búsqueda normalmente muy grandes.

Bibliografía

[1] Acid, S., y L. M. de Campos. «An algorithm for finding minimum d-separating sets in belief networks.» En *Conference of Uncertainty in Artificial Intelligence*. 1996, 3–10.

[2] Acid, S., L. M. de Campos, y J. F. Huete. «The search of causal orderings: A short cut for learning belief networks.» En *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. 2001, 216–227.

[3] Akaike, H. «New look at the statistical model identification.» *IEEE Transactions on Automatic Control*, 19, nº 6, (1974), 716–723.

[4] Alba, E. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[5] Alba, E., y M. Tomassini. «Parallelism and Evolutionary Algorithms.» *IEEE Transactions on Evolutionay Computation*, 6, nº 5, (2003), 443–462.

[6] Amir, E. «Efficient approximation for triangulation of minimum treewidth.» En *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*. Morgan Kaufmann, San Francisco, CA, 2001, 7–15.

[7] Andersen, S. K., K. G. Olesen, F. V. Jensen, y F. Jensen. «Hugin: A shell for building belief universes for expert systems.» En *Readings in Uncertain Reasoning* (G. Shafer, y J. Pearl, eds.), Morgan Kaufmann, 1990. 332–337.

[8] Arnborg, S., D. G. Corneil, y A. Proskurowski. «Complexity of finding embeddings in a k-tree.» *SIAM Journal of Algorithms and Discrete Methods*, 8, (1987), 277–284.

[9] Arnborg, S., J. Lagergren, y D. Seese. «Easy problems for tree-decomposable graphs.» *Journal of Algorithms*, 12, nº 2, (1991), 308–340.

[10] Backhouse, R. C. *Program Construction and Verification*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

[11] Balcázar, J. L. *Programación Metódica*. MacGraw-Hill, 1993.

[12] Baluja, S. «Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning.» *Informe Técnico. CMU-CS-94-163*, Pittsburgh, PA, 1994.

[13] —. «An empirical comparison of seven iterative and evolutionary function optimization heuristics.» *Informe Técnico. CMU-CS-95-193*, Pittsburgh, PA, 1995.

[14] Baluja, S., y S. Davies. «Using optimal dependency-trees for combinational optimization.» En *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, 30–38.

[15] Becker, A., y D. Geiger. «A sufficiently fast algorithm for finding close to optimal junction trees.» En *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence* (Morgan Kaufmann, ed.). 1996, 81–89.

[16] Beinlich, I., G. Suermondt, R. Chavez, y G. Cooper. «The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks.» En *Proceedings of the Second European Conference on AI and Medicine*. Springer-Verlag, 1989, 247–256.

[17] Bengoetxea, E. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. Tesis Doctoral, Ecole Nationale Supérieure des Télécommunications, Paris, France, December 2002.

[18] Bengoetxea, E., P. Larrañaga, I. Bloch, A. Perchant, y C. Boeres. «Learning and simulation of Bayesian networks applied to inexact graph matching.» *Pattern Recognition*, 35, nº 12, (2002), 2867–2880.

[19] Beyer, H. G., y H. P. Schwefel. «Evolution strategies: A comprehensive introduction.» *Journal Natural Computing*, 1, nº 1, (2002), 3–52.

[20] Blanco, R., I. Inza, y P. Larrañaga. «Learning Bayesian networks in the space of structures by estimation of distribution algorithms.» *International Journal of Intelligent Systems*, 18, (2003), 205–220.

[21] Bodlaender, H., A. Koster, F. van den Eijkhof, y L. van der Gaag. «Preprocessing for triangulation of probabilistic networks.» En *Uncertainty in Artificial Intelligence (UAI-01)*. Morgan Kaufmann, 2001, 32–39.

[22] Bonabeu, E., M. Dorigo, y G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Oxford, UK, 1999.

[23] de Bonet, J. S., C. L. Isbell, y P. Viola. «MIMIC. Finding optima by estimating probability densities.» En *Advances in Neural Information Processing Systems*. 1997, Volumen 9, 424–430.

[24] Bosman, P. A., y D. Thierens. «Linkage information processing in distribution estimation algorithms.» En *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99* (W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, y R. E. Smith, eds.). Morgan Kaufmann Publishers, San Francisco, LA, 1999, Volumen 1, 60–67.

[25] —. «Continuous iterated density estimation evolutionary algorithms within the IDEA framework.» En *Proceedings of the 2000*

Genetic and Evolutionary Computation Conference Workshop Program (A. S. Wu, ed.). 2000, 197–200.

[26] —. «Expanding from discrete to continuous estimation of distribution algorithms: The IDEA.» *Parallel Problem Solving from Nature - PPSN VI. Lecture Notes in Computer Science 1917*, (2000), 767–776.

[27] —. «IDEAs based on the normal kernels probability density function.» *Informe Técnico. UU-CS-2000-11*, Utrech University, 2000.

[28] —. «Negative log-likelihood and statistical hypothesis testing as the basis of model selection in IDEAs.» En *Genetic and Evolutionary Computation Conference GECCO-00. Late Breaking Papers.* Morgan Kaufmann, 2000, 51–58.

[29] Bouckaert, R. «A stratified simulation scheme for inference in Bayesian belief networks.» En *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94).* Morgan Kaufmann Publishers, San Francisco, CA, 1994, 110–117.

[30] Bouckaert, R. R. «Optimizing causal orderings for generating DAGs from data.» En *Uncertainty in Artificial Intelligence.* 1992, Volumen 8, 9–16.

[31] Box, G., y M. Muller. «A note on the generation of random normal deviates.» *Annals of Mathematical Statistics*, 29, (1958), 610–611.

[32] Buntime, W. «Theory refinement in Bayesian networks.» *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, (1991), 52–60.

[33] Butenhof, D. R. *Programming with POSIX Threads.* Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1997.

[34] de Campos, L. M., J. A. Gámez, P. Larrañaga, S. Moral, y T. Romero. «Partial abductive inference in Bayesian networks: an empirical comparison between GAs and EDAs.» En *Estimation*

of Distribution Algorithms. A New Tool for Evolutionary Computation (P. Larrañaga, y J. A. Lozano, eds.), Kluwer Academic Publishers, 2001. 319–337.

[35] DE CAMPOS, L. M., J. A. GÁMEZ, y S. MORAL. «Partial abductive inference in Bayesian belief networks using a genetic algorithm.» *Pattern Recognition Letters*, 20, nº 11-13, (1999), 1211–1217.

[36] DE CAMPOS, L. M., y J. M. PUERTA. «Stochastic local algorithm for learning belief networks: searching in the space of orderings.» En *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. 2001, 228–239.

[37] CANO, A., y S. MORAL. «Heuristic algorithms for the triangulation of graphs.» En *Information Processing and Management of Uncertainty in Konowledge-Based Systems*. París, 1994, 98–107.

[38] CANTU-PAZ, E. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.

[39] CASTILLO, E., J. GUTIERREZ, y A. HADI. *Expert Systems and Probabilistic Network Models*. Springer-Verlag, New York, 1997.

[40] CASTRILLO, P., y A. DÍEZ. *Formas Lógicas: Guía para el Estudio de la Lógica*. Universidad Nacional de Educación a Distancia, 2003.

[41] CHARNIAK, E., y S. E. SHIMONY. «Probabilistic semantics for rule based systems.» *Informe Técnico. CS-90-92*, Department of Computer Science. Brown University. Providence, Rhode Island, 1990.

[42] CHICKERING, D., D. GEIGER, y D. HECKERMAN. «Learning Bayesian networks is NP-hard.» *Informe Técnico. MSR-TR-94-17*, November 1994.

[43] CHOW, C. K., y C. N. LIU. «Approximating discrete probability distributions with dependence trees.» En *IEEE Transactions on Information Theory*. 1968, Volumen 14, 462–467.

[44] Cooper, G. F. «Nestor: A computer-based medical diagnostic aid that integrates causal and probabilistic knowledge.» *Informe Técnico. Hpp-84-48*, Stanford University, Stanford, 1989.

[45] Cooper, G. F., y E. Herskovits. «A Bayesian method for the induction of probabilistic networks from data.» *Machine Learning*, 9, (1992), 309–347.

[46] Corp., Norsys Software. «Netica Software Package.» http://www.norsys.com, 1998.

[47] Corporation, Scyld Computing. «Beowulf Parallel Wokstation Project.» http://www.beowulf.org, 1994.

[48] Cox, P., y T. Pietrzykowski. «Causes for events: Their computation and application.» En *Proceedings of CADE 86*. 1986, 608–621.

[49] Darwiche, A., y M. Hopkins. «Using recursive decomposition to construct elimination orders, jointrees, and dtrees.» En *ECSQARU '01: Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. Springer-Verlag, London, UK, 2001, 180–191.

[50] Darwin, C. *El Origen de las Especies*. Planeta - De Agostini S. A., 1992.

[51] Dawid, A. P. «Applications of a general propagation algorithm for probabilistic expert systems.» *Statistics and Computing*, 2, (1992), 25–36.

[52] Deaño, A. *Introducción a la Lógica Formal*. Aliaza Editorial S.A., 2002.

[53] DeGroot, M. H. *Optimal Statistical Decisions*. McGraw-Hill, New York, 1970.

[54] Dempster, A. «Covariance selection.» *Biometrics*, 28, (1972), 157–175.

[55] Dorigo, M., y G. Di Caro. «The ant colony optimization meta-heuristic.» En *New Ideas in Optimization* (D. Corne, M. Dorigo, y F. Glover, eds.), McGraw-Hill, London, 1999. 11–32.

[56] Dromey, G. *Program Derivation. The Development of Programs from Specifications.* Addison-Wesley, 1989.

[57] Eshelman, L. J., y J. D. Schaffer. «Productive recombination and propagating and preserving schemata.» En *Proceedings of the Third Workshop on Foundations of Genetic Algorithms.* Morgan Kaufmann, 1994, 299–313.

[58] Eshghi, K., y R. Kowalski. «Abduction through deduction.» *Informe técnico.*, Imperial College of Science and Technology, Department of Computing, 1988.

[59] Etxeberria, R., y P. Larrañaga. «Global optimization with Bayesian networks.» En *II Symposium on Artificial Intelligence. CIMAF99. Special Sesion on Distributions and Evolutionary Optimization.* 1999, 332–339.

[60] Flores, M. J. *Bayesian Networks Inference: Advanced Algorithms for Triangulation and Partial Abduction.* Tesis Doctoral, Universidad de Castilla-La Mancha, 2005.

[61] Forum, Message Passing Interface. «MPI: A message-passing interface standard.» *Informe Técnico. UT-CS-94-230*, 1994.

[62] Foster, I. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1995.

[63] Friedman, N., y D. Koller. «Being Bayesian about network structure. An Bayesian approach to structure discovery in Bayesian networks.» *Machine Learning*, 50, nº 1, (2003), 95–125.

[64] Fujisawa, T., y H. Orino. «An efficient algorithm of finding a minimal triangulation of a graph.» En *IEEE International Symposium on Circuits and Systems.* San Francisco, CA, 1974, 172–175.

[65] Fung, R., y B. Favero. «Backward simulation in Bayesian networks.» En *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*. Morgan Kaufmann Publishers, San Francisco, CA, 1994, 227–234.

[66] Fung, R., y C. Kuo-Chu. «Weighing and integrating evidence for stochastic simulation in Bayesian networks.» En *Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence (UAI-90)*. Elsevier Science Publishing Company, Inc., New York, NY, 1990, Volumen 5, 209–220.

[67] Gallagher, M. R. *Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling*. Tesis Doctoral, Department of Computer Science and Electrical Engineering, University of Queensland, 2000.

[68] Gallagher, M. R., M. Frean, y T. Downs. «Real-valued evolutionary optimization using a flexible probability density estimator.» En *Proceedings of Genetic and Evolutionary Computation Conference*. Morgan Kauffmann, 1999, 840–846.

[69] Gámez, J. A. *Inferencia Abductiva en Redes Causales*. Tesis Doctoral, Dpto de Ciencias de la Computación e Inteligencia Artificial. Escuela Técnica Superior de Ingeniería Informática. Universidad de Granada, junio 1998.

[70] Gámez, J. A., y J. M. Puerta. «Searching for the best elimination secuence in Bayesian networks by using Ant Colony based optimization.» *Pattern Recognition Letters*, 23, nº 1-3, (2002), 261–277.

[71] Geiger, D., y D. Heckerman. «Learning Gaussian networks.» *Informe Técnico. MSR-TR-94-10*, Redmond, WA, 1994.

[72] Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, y V. Sunderam. *PVM: Parallel Virtual Machine. A Users´ Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[73] Hansen, P., y N. Mladenovi. «An introduction to variable neighborhood search.» En *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, 1999, 433–458.

[74] Harik, G. R., F. G. Lobo, y D. E. Goldberg. «The compact genetic algorithm.» *IEEE Transactions on Evolutionary Computation*, 3, nº 4, (1999), 287–297.

[75] Heckerman, D. «A tutorial on learning with Bayesian networks.» *Informe Técnico. MSR-TR-95-06*, 1996.

[76] Heckerman, D., D. Geiger, y D. M. Chickering. «Learning Bayesian networks: The combination of knowledge and statistical data.» En *KDD Workshop*. 1994, 85–96.

[77] Henrion, M. «Propagating uncertainty in Bayesian networks by probabilistic logic sampling.» En *Uncertainty in Artificial Intelligence* (J. F. Lemmer, y L.Ñ. Kanal, eds.). North-Holland, Amsterdam, 1988, Volumen 2, 149–163.

[78] Howard, R., y J. Matheson. «Influence diagrams.» En *Readings on the Principles and Applications of Decision Analisys* (R. Howard, y J. Matheson, eds.). 1981, Volumen 2, 721–764.

[79] de Illarraza, A. Diaz, y P. Lucio. *Verificación de Programas y Metodología de la Programación*. Servicio Editorial Universidad del País Vasco, 1993.

[80] Jensen, F. V. *An Introduction to Bayesian Networks*. University College of London, 1996.

[81] Jensen, F. V., S. L. Lauritzen, y K. G. Olesen. «Bayesian updating in causal probalistic networks by local computations.» En *Computational Statistics Quaterly*. 1990, Volumen 4, 269–282.

[82] Jensen, F. V., K. G. Olesen, y S. K. Andersen. «An algebra of Bayesian belief universes for knowledge based systems.» En *Networks*. 1990, Volumen 20, 637–659.

[83] Kirpatrick, S., C. D. Gelatt, y M. P. Vecchi. «Optimization by simulated annealing.» *Science*, 220, (1983), 671–680.

[84] Kjaerulff, U. «Triangulation of graphs—Algorithms giving small total state space.» *Informe Técnico. R90-09*, Department of Computer Science, University of Aalborg, March 1990.

[85] —. «Optimal decomposition of probabilistic networks by simulated annealing.» *Statistics and Computing*, 2, (1992), 7–17.

[86] Kumar, U. P., y U. B. Desai. «Image interpretation using Bayesian networks.» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18, nº 1, (1996), 74–78.

[87] Larrañaga, P. *Aprendizaje Estructural y Descomposición de Redes Bayesianas via Algoritmos Genéticos.* Tesis Doctoral, Departamento de Ciencias de la Computación e Inteligencia Artificial. Euskal Herriko Unibertsitatea/Universidad del País Vasco, 1995.

[88] Larrañaga, P., R. Etxeberría, J. A. Lozano, y J. M. Peña. «Optimization by learning and simulation of Bayesian and Gaussian networks.» *Informe Técnico. EHU-KZAA-IK-4/99*, University of the Basque Country, 1999.

[89] Larrañaga, P., R. Etxeberria, J. A. Lozano, y J. M. Peña. «Combinatorial optimization by learning and simulation of Bayesian networks.» *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, (2000), 343–352.

[90] —. «Optimization in continuous domains learning and simulation on Gaussian networks.» *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, (2000), 201–204.

[91] Larrañaga, P., C. M. H. Kuijpers, R. H. Murga, y Y. Yurramendi. «Learning Bayesian nework structures by searching for the best ordering with genetic algorithms.» En *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans.* 1996, Volumen 26–4, 487–493.

[92] Larrañaga, P., y J. A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[93] Larrañaga, P., J. A. Lozano, y E. Bengoetxea. «Estimation of distribution algorithms based on multivariate normal and Gaussian networks.» *Informe Técnico. KZZA-IK-1-01*, Department of Computer Science and Artificial Intelligence. University of the Basque Country, 2001.

[94] Larranaga, P., C. Kuijpers, M. Poza, y R. Murga. «Decomposing Bayesian networks: Triangulation of the moral graph with genetic algorithms.» *Statistics and Computing*, 7, nº 1, (1997), 19–34.

[95] Lauritzen, S. L. *Graphical Models*. Nº 17 en Oxford Statistical Science Series. Clarendon Press, Oxford, 1996.

[96] Lauritzen, S. L., T. P. Speed, y K. Vijayan. «Decomposable graphs and hypergraphs.» *Journal of the Australian Mathematical Society, Series A*, 36, (1984), 12–29.

[97] Lauritzen, S. L., y D. J. Spiegelhalter. «Local computations with probabilities on graphical structures and their application on expert systems.» *Journal of the Royal Statistical Society B*, 50, nº 2, (1988), 157–224.

[98] Leighton, T., y S. Rao. «Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms.» *Journal of the ACM*, 46, nº 6, (1999), 787–832.

[99] Li, Z., y B. D´Ambrosio. «An efficient approach for finding the MPE in belief networks.» En *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence*. Morgan Kauffmann, 1993, 342–349.

[100] Lozano, J. A., R. Sagarna, y P. Larrañaga. «Parallel estimation of distribution algorithms.» *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, (2001), 125–142.

[101] Lozano, J.A., P. Larrañaga, I. Inza, y E. Bengoetxea. *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*, Volumen 192 de *Studies in Fuzziness and Soft Computing*. Springer, 2006.

[102] Madsen, A. L., y F. V. Jensen. «Lazy propagation in junction trees.» En *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*. 1998, 362–369.

[103] —. «Lazy propagation: a juntion tree inference algorithm based on lazy evaluation.» *Artificial Intelligence*, 113, (1999), 203–245.

[104] Meila, M., y M. I. Jordan. «Triangulation by continuous embedding.» *Informe Técnico. AIM-1605*, 1997.

[105] Mendiburu, A., J. A. Lozano, y J. M. Alonso. «Parallel implementation of EDAs based on probabilistic graphical models.» *IEEE Transactions on Evolutionary Computation*, 9, nº 4, (2005), 406–423.

[106] Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1992.

[107] Mühlenbein, H. «The equation for response to selection and its use for prediction.» En *Evolutionary Computation*. 1998, Volumen 5, 303–346.

[108] Mühlenbein, H., y T. Mahnig. «FDA: A scalable evolutionary algorithm for the optimization of additively decomposed functions.» *Evolutionary Computation*, 7, nº 4, (1999), 353–376.

[109] Mühlenbein, H., T. Mahnig, y A. Rodriguez. «Schemata, distributions and graphical models in evolutionary optimization.» *Journal of Heuristics*, 5, (1999), 215–247.

[110] Mühlenbein, H., y H. Voigt. «Gene pool recombination in genetic algorithms.» En *Metaheuristics: Theory and Applications* (J. P. Kelly, y I. H. Osman, eds.). Kluwer Academic Publishers, 1996, 53–62.

[111] Murphy, P. M., y D. W. Aha. «UCI Repository of Machine Learning Databases.» http://www.ics.uci.edu/ mlearn/MLRepository.html, 1995.

[112] Neapolitan, R. E. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[113] Nilsson, D. «An efficient algorithm for finding the m most probable configurations in Bayesian networks.» *Statistics and Computing*, 8, nº 2, (1998), 159–173.

[114] Ocenásek, J., y J. Schwarz. «The distributed Bayesian optimization algorithm for combinatorial optimization.» En *EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*. 2001, 115–120.

[115] de la Ossa, L., J. A. Gámez, y J. M. Puerta. «Initial approaches to the application of islands-based parallel EDAs in continuous domains.» En *Parallel processing, 2005. ICPP 2005 workshops*. 2005, 580–587.

[116] Pearl, J. «Distributed revision of composite beliefs.» *Artificial Intelligence*, 33, nº 2, (1987), 173–215.

[117] —. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[118] Pelikan, M., y D. E. Goldberg. «Genetic algorithms, clustering and the breaking of the symmetry.» *Parallel Problem Solving from Nature - PPSN VI. Lecture Notes in Computer Science 1917*, (2000), 385–394.

[119] Pelikan, M., D. E. Goldberg, y E. Cantú-Paz. «BOA: The Bayesian optimization algorithm.» *Proceedings of the Genetic and Evolutionary Computation Conference GECCO99*, 1, (1999), 525–532.

[120] Pelikan, M., D. E. Goldberg, y K. Sastry. «Bayesian optimization algorithm, decision graphs and Occam razor.» *Informe Técnico. 200020*, Illinois Genetic Algorithms Laboratory. University of Illinois at Urbana-Champaing., 2000.

[121] Peng, Y. *A Formalization of Parsimonious Covering and Probabilistic Reasoning in Abductive Diagnostic Inference*. Tesis Doctoral, Department of Computer Science, University of Maryland, 1986.

[122] Puerta, J. M. *Métodos Locales y Distribuidos para la Construcción de Redes de Creeencia Estáticas y Dinámicas*. Tesis Doctoral, E. T. S de Ingeniería Informática. Universidad de Granada, 2001.

[123] Rebane, G., y J. Pearl. «The recovery of causal polytrees from statistical data.» En *Uncertainty in Artificial Intelligence*. 1989, Volumen 3, 175–182.

[124] Ripley, B. D. *Stochastic Simulation*. John Willey and Sons, 1987.

[125] Robertson, N., y P. D. Seymour. «Graph minors XIII. The disjoint paths problem.» *Journal of Combinatorial Theory, Series B*, 63, (1995), 65–110.

[126] Robinson, R. «Counting unlabelled acyclic digraphs.» En *Lecture Notes in Mathematics: Combinatorial Mathematics V*. Springer-Verlag, 1977, 28–43.

[127] Romero, T., y P. Larrañaga. «Triangulation of Bayesian networks with estimation of distribution algorithms.» *International Journal of Approximate Reasoning*. Pendiente de revisión.

[128] Romero, T., P. Larrañaga, y B. Sierra. «Learning Bayesian networks in the space of orderings with estimation of distribution algorithms.» *International Journal of Pattern Recognition and Artificial Intelligence*, 18, (2004), 607–625.

[129] Rose, D. J. «A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations.» *Graph Theory and Computing*, (1973), 183–217.

[130] Rose, D. J., R. E. TArjan, y G. S. Lueker. «Algorithmic aspects of vertex elimination on graphs.» *SIAM Journal of Computation*, 5, (1976), 266–283.

[131] Santana, R., y A. Ochoa. «Dealing with constraints with estimation of distribution algorithms: The univariate case.» *Second Simposium on Artificial Intelligence. Adaptative Systems. CIMAF 99*, (1999), 378–384.

[132] Santana, R., Fr. B. Pereira, E. Costa, A. Ochoa-Rodriguez, P. Machado, A. Cardoso, y M. Soto. «Probabilistic evolution and the busy beaver problem.» En *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference* (D. Whitley, ed.). Las Vegas, Nevada, USA, 2000, 261–268.

[133] Santos, E. Jr., y E. S. Santos. «Polynomial solvability of cost-based abduction.» *Artificial Intelligence*, 86, nº 1, (1996), 157–170.

[134] Schwarz, G. «Estimating the dDimension of a model.» *The Annals of Statistics*, 6, nº 2, (1978), 461–464.

[135] Sebag, M., y A. Ducolombier. «Extending population-based incremental learning to continuous search spaces.» *Parallel Problem Solving from Nature - PPSN V*, (1998), 418–427.

[136] Seroussi, B., y J. L. Goldmard. «An algorithm directly finding the k most probable configurations in Bayesian networks.» En *International Journal of Approximate Reasoning*. 1994, 11, 205–233.

[137] Shachter, R., y M. Peot. «Simulation approaches to general probabilistic inference on belief networks.» En *Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence (UAI-90)*. Elsevier Science Publishing Company, Inc., New York, NY, 1990, Volumen 5, 221–234.

[138] Shachter, R. S., y C. R. Kenley. «Gaussian influence diagrams.» *Management Science*, 35, nº 5, (1989), 527–550.

[139] Shanahan, M. «Prediction is deduction but explanation is abduction.» En *Proceedings of the 11th International Conference on Artificial Intelligence*. 1989, 1055–1060.

[140] Shenoy, P. P., y G. R. Shafer. «Axioms for probability and belief-function propagation.» *Readings in Uncertain Reasoning*, (1990), 575–610.

[141] Shimony, S. E. «On irrelevance and partial assignments to belief networks.» *Informe Técnico. CS-90-14*, Department of Computer Science. Brown University. Providence, Rhode Island, 1990.

[142] —. «A probabilistic framework for explanation.» *Informe Técnico. CS-91-57*, Department of Computer Science. Brown University. Providence, Rhode Island, 1991.

[143] Soto, M., A. Ochoa, S. Acid, y L. M. de Campos. «Introducing the polytree aproximation of distribution algorithm.» *Second Simposium on Artificial Intelligence. Adaptative Systems. CIMAF 99*, (1999), 360–367.

[144] Speed, T., y H. Kiiveri. «Gaussian markov distributions over finite graphs.» *Annals of Statistics*, 14, (1986), 138–150.

[145] Srinivas, S., y P. Nayak. «Efficient enumeration of instantiations in Bayesian networks.» En *Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*. Portland, Oregon, 1996, 500–508.

[146] Stickel, M. E. «A prolog-like inference system for computing minimum-cost abductive explanations in natural language interpretation.» *Informe Técnico. 451*, AI Center, SRI International, 1988.

[147] Syswerda, G. «Simulated crossover in genetic algorithms.» En *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*. Morgan Kaufmann, 1993, 239–255.

[148] Tanenbaum, A. S. *Structured Computer Organization*. Prentice-Hall Inc., 1984.

[149] Tarjan, R. E., y M. Yannakakis. «Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs.» *SIAM Journal of Computation*, 13, nº 3, (1984), 566–579.

[150] Wen, W. X. «Optimal decomposition of belief networks.» En *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI-90)*. Elsevier Science Inc., New York, NY, USA, 1990, 209–224.

[151] —. «Optimal decomposition of belief networks.» En *Uncertainty in Artificial Intelligence* (P. P. Bonissone, M. Henrion, L. N. Kanal, y J. F. Lemmer, eds.). North-Holland, Amsterdam, 1991, Volumen 6, 209–224.

[152] Wermuth, N. «Model search among multiplicative models.» *Biometrics*, 32, (1976), 253–263.

[153] Whitley, D. «The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best.» En *Proceedings of the 3rd International Conference on Genetic Algorithms*. 1989, 116–121.

[154] Whittaker, J. *Graphical Models in Applied Multivariate Statistics*. John Wiley and Sons, 1990.

[155] Wong, S. K. M, D. Wu, y C. J. Butz. «Triangulation of Bayesian networks: A relational database perspective.» En *TSCTC '02: Proceedings of the Third International Conference on Rough Sets and Current Trends in Computing*. Springer-Verlag, London, UK, 2002, 389–396.

[156] Yannakakis, M. «Computing the minimal fill-in is NP-complete.» *Journal of Algebraic Discrete Methods*, 2, (1981), 77–79.

Citation index