eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Departamento de Arquitectura y Tecnología de Computadores

Avances en Algoritmos de Estimación de Distribuciones. Alternativas en el Aprendizaje y Representación de Problemas

Tesis que presenta

Teresa Miquélez Echegaray

Para obtener el grado de

Doctora en Informática

Directores : Pedro Larrañaga Múgica
Endika Bengoetxea Castro

Enero 2010

Agradecimientos

Desde estas líneas quiero expresar mi más sincero agradecimiento a todas las personas que, desde diferentes aspectos, me han acompañado en esta interesante experiencia.

En primer lugar y muy especialmente quiero dar las gracias a los doctores Pedro Larrañaga y Endika Bengoetxea, directores de esta tesis, por ofrecerme la posibilidad de realizar esta tesis, por su aportación científica, por su confianza y por el tiempo y paciencia que me han dedicado en todo momento.

También quiero agradecer la gran ayuda que me ha proporcionado Alex Mendiburu por su presencia incondicional y sus valiosas sugerencias y comentarios durante el desarrollo de este trabajo. Gracias Alex.

He tenido la suerte de poder formar parte del grupo de investigación *Intelligent Systems Group*, un grupo cercano formado por personas estupendas con las que es un placer poder trabajar. Quiero dar las gracias a Pedro Larrañaga, José Antonio Lozano, Iñaki Inza, Endika Bengoetxea, Alex Mendiburu y José Miguel por todo su esfuerzo en mantener un grupo como éste y al resto de compañeros y ex-compañeros del grupo, en especial a Rosa Blanco, Roberto Santana, Dinora Morales, Rubén Armañanzas, Guzmán Santafé, Txomin Romero, Ramón Sagarna, Aritz Pérez, Juan Diego Rodríguez, Borja Calvo, Carlos Echegoyen, Jose Antonio Fernandes, Jose Luis Flores por todo lo compartido: trabajo, charlas y buenos momentos.

No puedo olvidar el apoyo y colaboración de mis compañeros de Departamento, especialmente de aquellos con los que he compartido docencia durante estos años. También a ellos quiero darles las gracias.

En el plano personal, quiero agradecer a mis hijos su cariño y comprensión en todo momento. A todos los amigos que han estado ahí, que me han escuchando y animando durante todo este tiempo, quiero agradecerles la confianza y paciencia admirables que han demostrando. Muchas gracias a todos.

Resumen

La Computación Evolutiva es una disciplina que se caracteriza por crear un conjunto de posibles soluciones y hacerlas evolucionar generación a generación con el propósito de resolver problemas de optimización. Ejemplos de paradigmas de computación evolutiva ampliamente reconocidos son los Algoritmos Genéticos (*Genetic Algorithms*, GAs) y los Algoritmos de Estimación de las Distribuciones (*Estimation Distribution Algorithms*, EDAs). La principal diferencia entre estos modelos está en la forma de mejorar el conjunto de soluciones en cada generación. En los GAs, la evolución se basa en la utilización de operadores de cruce y mutación, sin expresar explícitamente las características de los individuos seleccionados dentro de una población. Los EDAs tienen en cuenta estas características explícitas al considerar las interdependencias entre las diferentes variables que representan a un individuo y aprender el modelo gráfico probabilístico que las representa.

Aunque en la mayoría de los problemas de optimización los EDAs obtienen muy buenos resultados, es posible encontrar aspectos de estos algoritmos susceptibles de mejora con el fin de aumentar el rendimiento de este método. En este sentido, cabe destacar la utilización del valor obtenido por cada individuo al ser evaluado. Mediante la función de evaluación o función objetivo, a cada individuo se le calcula un valor que indica cómo de próximo se encuentra ese individuo del óptimo. Los EDAs utilizan este valor para seleccionar a aquellos individuos que se utilizarán para crear el modelo gráfico probabilístico. Sin embargo, los EDAs no utilizan toda la información que proporciona la función objetivo. Es decir, a la hora de obtener el modelo gráfico probabilístico, solo tienen en cuenta el valor de las variables predictoras de los individuos seleccionados, considerándolos todos iguales. Esto provoca que individuos con diferente valor de función objetivo sean, si han sido seleccionados, considerados iguales por los EDAs a la hora de obtener el modelo que se utilizará para generar nuevos individuos. Este paso es fundamental para la convergencia del proceso, ya que el modelo gráfico probabilístico es la forma que utilizan los EDAs para representar la información de los datos analizados.

También es importante analizar la manera más apropiada de representación de un problema para su resolución mediante EDAs. Aunque este aspecto está directamente ligado a cada problema particular, siempre es posible encontrar diferentes formas de representar las características que mejor definan las diferencias entre posibles soluciones. Elegir una u otra forma de afrontar el problema puede resultar muy importante a la hora de resolverlo. Además, cada representación necesita su función de evaluación correspondiente, lo cual nos lleva a elegir la función objetivo más adecuada para una mejor y más rápida convergencia del proceso.

En esta tesis se analiza la importancia de estos dos aspectos. Pensando en añadir información sobre la función objetivo en el proceso de aprendizaje, se ha desarrollado un nuevo método de optimización que funciona de manera similar a los EDAs, pero que utiliza clasificadores Bayesianos en el proceso de evolución. De esta forma, los

individuos se clasifican según el valor obtenido en la función objetivo y utilizamos esta información para, a través de clasificadores Bayesianos, obtener el modelo gráfico probabilístico. A este nuevo método se le ha llamado Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOA).

Para analizar la importancia que puede tener, en el proceso de evolución de los EDAs, utilizar una determinada forma de representación del problema, es necesario trabajar con un ejemplo concreto que resulte realmente completo y que pueda utilizarse para formalizar gran cantidad de casos prácticos. Teniendo en cuenta esto, nos hemos centrado en el problema de la satisfactibilidad (SAT). Este problema es uno de los más importantes en la teoría computacional, ya que representa un modelo genérico mediante el cual se pueden formalizar gran cantidad de casos prácticos en diferentes campos de investigación, sobre todo problemas de decisión tales como diseño, síntesis y verificación de circuitos integrados, optimización y planificación de tareas. Además, este problema es NP-completo. La mayor parte de los trabajos realizados para resolver el SAT están basados en métodos exactos y completos. Últimamente también se han utilizado algoritmos evolutivos en su resolución, principalmente algoritmos genéticos. Basándonos en los trabajos realizados en esta línea, se analizan nuevas formas de representación, buscando en cada caso, la función objetivo adecuada y estudiando el comportamiento de los EDAs en los diferentes casos.

Laburpena

Konputazio ebolutiboa optimizazio diziplina bat da non optimizazio problemak ebazteko soluzio posibleen azpimultzo baten bidez hurrengo soluzioen generazio bat sortzen den. Ezagunen diren teknikak arlo honen barruan Algoritmo genetikoak (*Genetic Algorithms*, GAs) eta *Estimation Distribution Algorithms* (EDA) izenekoak ditugu. Bien arteko desberdintasun nagusiena eboluzionatzeko prozeduran datza, non GAetan gurutzaketa eta mutazio tekniketan oinarritzen den eta EDAetan eredu grafiko probabilistikoetan oinarritutako tekniketan.

Nahiz EDAk hainbat optimizazio problemetan emaitza onak lortzen dituztela frogatu bada ere, hobekuntzarako hainbat esparru existitzen dira. Adibidez, EDAen kasuan populazio bakoitzean eboluziorako aukeratzen diren indibiduoen aldagaien balioak kontsideratzen dira, baina indibiduoen arteko *fitness* edo soluzioaren egokitasun maila ez da normalean kontuan hartzen eta indibiduo guztiak berdin bezala tratatzen dira eredu probabilistikoa sortzeko orduan. Ezaugarri hau inportantea da eredu hau erabiltzen baita aldagai guztien arteko menpekotasunak irudikatzeko, hau izanik eboluzioa bideratzeko mekanismo nagusiena.

Bestalde, EDAen eraginkortasunari dagokionez, optimizazio problemaren adierazpen modua oso inportantea da. Ezaugarri hau optimizazio problema bakoitzeko desberdina bada ere, egokiena da aukeratutako problemaren adierazpideak hoberen erakustea soluzio posible desberdinen arteko desberdintasun nagusienak. Problemaren

indibiduoen adierazpenaz gain, egokitasun funtzioaren definizio egokiena ere aukeratu beharra dago eboluzio prozedura erabakigarria izan dadin.

Tesi honetan bi ezaugarri hauen garrantzia aztertzen da EDAen eraginkortasun orokorrean duten eragina hobetzeko. Egokitze funtzioaren balioaren esangarritasuna hobetzeko helburuari dagokionez, tesiaren emaitzetako bat izan da EDAen antzeko optimizazio paradigma berri bat definitu izatea, problemaren ikasketa fasea aldatuz. Gure proposamen berri honen arabera, populazio bakoitzeko indibiduoak beren egokitze funtzioaren balioaren arabera sailkatzen dira, eta orden hau jarraituz klasifikatzaile Bayesiar formako eredu grafiko probabilistiko bat eraikitzen da. Teknika berri honi Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOA) deitu diogu.

Azkenik, indibiduoen adierazpenaren eragina aztertzeko asmoz, EDAen eraginkortasun orokorrean dagokionez, optimizazio problema konkretu bat aukeratu dugu, SAT izenekoa (*satisfiability*). Hau konputazio teoriaren barnean oso ezaguna den eta NP-osoa den optimizazio problema multzo bat da, zeinak problemen adierazpen eredu orokor bat aurkezten duen ikerketa kasu praktikoei erantzun bat eman ahal izateko, hots, erabakiak hartzeko problemak, zirkuitu integratuen diseinu eta frogaketa, optimizazioa eta atazen planifikazioa. Publikatuta dauden SATerako konputazio ebolutiboaren tekniken ikerketa lanetan oinarrituz, SAT optimizazio problemen indibiduo adierazpen desberdinak aztertu ditugu tesi honetan, eta hauekin batera egokien zaizkien fitness funtzio optimoenak zein diren aztertuz EDAentzako eraginkortasun hoberena lortzeko asmoz.

Summary

Evolutionary computation is a discipline characterised by creating a set of possible solutions and to make it evolve generation after generation in order to solve optimization problems. Examples of broadly known evolutionary computation paradigms are Genetic Algorithms (GAs) and Estimation Distribution Algorithms (EDAs). The main difference between these models is the evolution process: In GAs this is based on crossover and mutation operators, without explicit expression of the characteristics of selected individuals within the population. EDAs take these explicit characteristics into account by considering the interdependencies between the variables that form an individual and by learning a probabilistic graphical model that represents them.

Even if EDAs show good results in many optimization problems, there are many aspects for improvement in order to enhance their performance. One of such aspects is to take into consideration the fitness of each individual when it is evaluated using the fitness function, which assigns a value to each individual expressing how close it is from the optimum. EDAs use this value to select those individuals that will be considered for creating the probabilistic graphical model. However, EDAs only consider the values of predictor variables of selected values, considering them all as equally valid when

learning the model to generate the new population. This step is essential since the probabilistic graphical model is used by EDAs to represent all dependencies between values.

Secondly, it is important to analyse the most appropriate way to represent a problem for its solving by EDAs. Even if this aspect is fully dependent on each particular optimization problem, the best choice is to try to best show the differences between the different solutions. The way in which the problem is represented has an important influence on the performance of EDAs. Furthermore, each individual representation of the problem requires a corresponding fitness function to be defined, which leads us to the need to choose the most appropriated fitness function for a better convergence process.

This thesis analyses the importance of these two aspects in the overall performance. With the aim of adding more information to the fitness function in the learning process, we have developed a new optimization method that works in a similar way as EDAs but using Bayesian classifiers for the learning step. Following this, individuals are classified according to their fitness and this information is used for building a probabilistic graphical model in the form of a Bayesian classifier. We call this new method Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOA).

Finally, in order to study the importance of a concrete individual representation in EDA's performance, we choose as a concrete optimization problem the satisfiability one (SAT). This NP-complete problem is broadly known in computational theory since it represents a generic model through which diverse research practical cases can be formalised using a generic representation, mainly decision making problems such as synthesis and verification of integrated circuits, optimization and task planning. Based on published works using evolutionary computation techniques for SAT, different representation ways are presented and analysed in order to investigate which is the best problem representation and fitness function to improve the performance of EDAs.

Publicaciones de esta tesis doctoral

Durante el desarrollo de esta tesis doctoral se han publicado varios trabajos en revistas, libros y congresos, sobre todo a nivel internacional. La lista completa de publicaciones es la siguiente:

- Miquélez, T., Bengoetxea, E., Mendiburu, A., Larrañaga, P.(2007): Combining Bayesian classifiers and estimation of distribution algorithms for optimization in continuous domains. *Connection Science*. 19 (4), 297-319.

- Miquélez, T., Bengoetxea, E., Larrañaga, P.(2006): Evolutionary Bayesian classifier based optimization in continuous domains. Lecture Notes in Computer Science 4247, *Proceedings of the 6th International Conference Simulated Evolution and Learning.*–SEAL´06. T.-D Wang et al. (Eds). 529-536.

- Miquélez, T., Bengoetxea, E., Larrañaga, P.(2006): Bayesian classifiers in optimization: An EDA-like approach. *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, Lozano, J.A.; Larrañaga, P.; Inza, I.; Bengoetxea, E. (Eds.). Springer. 221-242.

- Miquélez, T., Pérez-Beltrán, O., Bengoetxea, E., Larrañaga, P.(2005): Optimización mediante clasificadores Bayesianos evolutivos en entornos continuos. *CEDI 2005, IV Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, Spain.

- Miquélez, T.,Bengoetxea, E., Larrañaga, P.(2004): Applying Bayesian classifiers to Evolutionary Computation. Technical report, KAT-IK-04-01 of the Department of Computer Architecture and Technology, University of the Basque Country, Spain.

- Miquélez T., Bengoetxea, E., and Larrañaga, P. (2004): Evolutionary computation based on Bayesian classifiers. *International Journal of Applied Mathematics and Computer Science*, 14(3), 335-349.

- Bengoetxea, E., Miquélez, T., Lozano, J. A., and Larrañaga, P. (2001): Experimental results in function optimization with EDAs in continuous domain. *Estimation of Distribution Algorithms. A new tool for Evolutionary Computation*, Larrañaga, P. and Lozano, J.A. (eds.). Kluwer Academic Publishers.

- Miquélez, T., Bengoetxea, E., Morlán, I., and Larrañaga, P. (2001): Obtención de filtros para restauración de imágenes por medio de algoritmos de estimación de distribuciones. *CAEPIA 2001, Conferencia de la Asociación Española para la Inteligencia Artificial.*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Dentro de las técnicas de computación evolutiva, *los Algoritmos de Estimación de las Distribuciones* (EDAs) se han hecho un hueco importante debido a que se apoyan en la teoría de la probabilidad, lo cual les da una robustez que no tienen otras técnicas basadas principalmente en heurísticos con un componente aleatorio importante.

Al igual que los *Algoritmos Genéticos* (GAs), los EDAs están basados en poblaciones que evolucionan. Sin embargo, se diferencian de los GAs, en que en los EDAs la evolución de las poblaciones no se lleva a cabo por medio de los operadores de cruce y mutación. En lugar de ello, la nueva población de individuos se muestrea a partir de un modelo gráfico probabilístico. Este modelo gráfico probabilístico representa una factorización de la distribución de probabilidad estimada a partir del conjunto de individuos seleccionados de entre los que constituyen la generación anterior. A través de esta distribución de probabilidad, los EDAs tienen en cuenta las interrelaciones entre las variables representadas por los individuos seleccionados en cada generación.

Uno de los aspectos más complicados de todo este proceso consiste precisamente en estimar dicha distribución de probabilidad. Es decir, en aprender el modelo gráfico probabilístico que represente las relaciones entre las variables de los individuos seleccionados. Como veremos en el capítulo 3, en el que presentamos en detalle estos algoritmos, podemos encontrar diferentes tipos de EDAs que se diferencian entre si en la forma en que aprenden dicho modelo gráfico probabilístico y en el número de relaciones entre variables que tienen en cuenta en este aprendizaje.

En cualquier caso, la obtención del modelo se hace en base a los individuos seleccionados, por tanto, es fundamental considerar cómo vamos a realizar esa selección. Por otro lado, todo este proceso se basa en el valor que obtiene cada individuo al ser evaluado. La elección de una función de evaluación apropiada es fundamental para un buen rendimiento de estos algoritmos. Además, esta función de evaluación está relacionada con la forma en que se ha decidido representar cada posible solución. Todo esto son aspectos importantes que influyen en el resultado final y que merece la pena analizar.

1.1 Motivación

El aprendizaje del modelo gráfico probabilístico tiene una importancia decisiva en el proceso evolutivo ya que representa las características de los mejores individuos de cada generación y se utiliza para muestrear los nuevos individuos de la generación siguiente. Para realizar este aprendizaje se selecciona a los mejores, de entre todos los individuos de la población, en cada generación. Sobre la influencia que tiene la forma en que se realiza la selección se han llevado a cabo diferentes investigaciones (Santana, 2003; Lima y col., 2007). Sin embargo, una vez realizada esta selección no se establece ninguna diferencia entre los individuos seleccionados, considerando a todos iguales. En este sentido, la principal motivación que nos impulsó a comenzar este trabajo fue la necesidad de añadir algún tipo de información adicional que diferenciara a unos individuos de otros, aportando más información al modelo. Para ello, antes de realizar la selección, se clasifica a todos los individuos de la población según el valor obtenido por cada individuo al ser evaluado. De esta forma, a cada individuo se le añade una nueva variable: la clase que se le ha asignado en esta clasificación. Esto permite seleccionar, no sólo a los mejores, sino incluir individuos que abarquen otras franjas de valores cuyas características puedan aportar información relevante al proceso evolutivo. A continuación, para el aprendizaje del modelo gráfico probabilístico, se utilizan clasificadores Bayesianos en los que se incluye la nueva variable clase.

Otro aspecto importante que se ha querido analizar, en relación a un mejor rendimiento de los EDAs, es la forma en que representamos la información que se va a manejar en todo el proceso evolutivo. Los algoritmos evolutivos en general y los EDAs en particular, trabajan teniendo en cuenta el valor obtenido por los individuos en cada generación. Este valor depende de la forma en que se plantee la representación del problema concreto a resolver y de la función objetivo utilizada. Tanto la elección de las variables que forman el individuo, como la función objetivo que se utilice, son factores decisivos que condicionan todo el proceso. Teniendo en cuenta esto, parece importante poner especial cuidado en la elección de la representación de individuos que se va a utilizar, así como en la función con la cual los vamos a evaluar. Sin embargo, ambos aspectos están estrechamente relacionados entre si y dependen del ejemplo concreto que vayamos a resolver. Por lo tanto, es necesario analizarlos tomando como referencia un problema de optimización concreto.

1.2 Principales contribuciones de este trabajo

Como principales contribuciones de esta tesis presentamos un conjunto de algoritmos evolutivos, similares a los EDAS, que se diferencian de estos en que utilizan clasificadores Bayesianos para aprender el modelo gráfico probabilístico que representa las características de los individuos seleccionados. Para ello, se clasifica a los individuos atendiendo al valor obtenido por estos en la función objetivo. A estos algoritmos

los identificamos como Algoritmos Evolutivos de Optimización Basados en Clasificadores Bayesianos (*Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOAs)*)

Además, basándonos en un problema de optimización complejo, presentamos nuevas formas de afrontar la resolución de un problema. Para cada propuesta, buscamos la función objetivo más apropiada y analizamos el comportamiento de los EDAs en cada caso. Este es un tema que no se trata normalmente en la literatura relacionada con los EDAs. La elección de las variables que componen el individuo, así como la función de evaluación, están relacionas directamente con el problema que queremos resolver. Normalmente esta decisión se deja en manos de un experto en el problema, como algo externo a los propios EDAs. Sin embargo, uniendo ese conocimiento del experto a nuestra experiencia con los EDAs podemos plantear nuevas opciones, más eficientes, de representación del problema. En este trabajo analizamos todo esto utilizando para ello uno de los problemas más importantes dentro de la teoría computacional, el problema de la Satisfactibilidad (SAT).

1.3 Estructura de esta memoria

Esta memoria se ha estructurado de la siguiente forma: En este primer capítulo se ha presentado el trabajo realizado en esta tesis, comentando la motivación inicial y las principales contribuciones realizadas. En el capítulo 2 presentamos los modelos gráficos probabilísticos que utilizan los EDAs, ya que constituyen una parte importante de estos algoritmos. A continuación, se presenta una revisión de los Algoritmos de Estimación de las Distribuciones en el capítulo 3. El capítulo 4 está dedicado a presentar el nuevo conjunto de Algoritmos Evolutivos de Optimización basados en Clasificadores Bayesianos (*Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOAs)*) comentado anteriormente como una de las principales contribuciones de este trabajo. Las diferentes posibilidades de representar un problema para su resolución mediante EDAs se presenta en el capítulo 5, buscando en cada caso la función objetivo apropiada y comprobando las ventajas de dicha opción. En el último capítulo se extraen las conclusiones más generales según los resultados obtenidos en los capítulos anteriores y se proponen posibles trabajos a realizar en el futuro que profundicen en el trabajo realizado.

Capítulo 2

Modelos Gráficos Probabilísticos

En este capítulo introducimos dos modelos gráficos probabilísticos que se utilizan para representar la distribución de probabilidad de los individuos seleccionados en los algoritmos de estimación de las distribuciones: las redes Bayesianas y las redes Gaussianas. Para ambos casos, presentamos diferentes algoritmos para extraer las características subyacentes de los datos y algunos métodos para simular dichos modelos.

2.1 Introducción

Los modelos gráficos probabilísticos (*PGM, probabilistic graphical models*) se han convertido en uno de los paradigmas más utilizados para codificar conocimiento incierto (Howard y Matheson, 1981; Lauritzen, 1996; Pearl, 1988). Si consideramos un modelo como la descripción matemática de un fenómeno, los modelos probabilísticos describen el fenómeno en términos de comportamiento probabilístico de los procesos. Un modelo gráfico probabilístico se compone de una estructura gráfica y un conjunto de distribuciones de probabilidad locales referidas a la estructura. Un modelo gráfico probabilístico se representa mediante un grafo en el que cada nodo es una variable y las aristas entre los nodos representan las dependencias condicionales entre las variables. Si la estructura de la red es un modelo gráfico acíclico dirigido, el modelo gráfico representa una factorización de la probabilidad conjunta de todas las variables aleatorias. Esto es, la probabilidad conjunta se factoriza como un producto de distribuciones condicionales. La estructura del grafo indica las dependencias directas entre las variables aleatorias. Este tipo de modelo gráfico es conocido como un modelo gráfico dirigido o red Bayesiana. Si todas las variables toman valores continuos, la función de densidad conjunta sigue una densidad Gaussiana multivariante y en este caso hablaremos de red Gaussiana.

En este capítulo se presentan las redes Bayesianas como un modelo gráfico probabilístico (Pearl, 1985) ampliamente utilizado para la representación gráfica de la incertidumbre en el conocimiento de sistemas expertos (Heckerman y Wellman, 1995).

Dedicaremos especial interés a los dos aspectos más importantes en la implementación de los EDAs, el aprendizaje de la estructura de la red a partir de los datos y su simulación. A continuación, utilizando el mismo esquema, introducimos las redes Gaussianas como un ejemplo de modelo gráfico probabilístico en el que se asume que la función de densidad conjunta sigue una densidad Gaussiana multivariante (Whittaker, 1990).

2.2 Redes Bayesianas

Una red Bayesiana es un modelo gráfico acíclico dirigido más una distribución de probabilidad sobre sus variables. Por tanto, una red Bayesiana está formada por dos partes: una parte cualitativa que expresa las relaciones de independencia condicional entre las variables y una parte cuantitativa compuesta por los valores del conjunto de distribuciones de probabilidad condicional. La topología o estructura de la red nos da información sobre las dependencias probabilísticas entre las variables, pero también sobre las dependencias condicionales de dos variables (o dos conjuntos de variables) dada otra (u otras) variable(s). Las redes Bayesianas constituyen una manera práctica y compacta de representar el conocimiento.

Sea $\boldsymbol{X} = (X_1, \ldots, X_n)$ un conjunto de variables aleatorias, donde x_i es un valor de X_i, la i-ésima componente de $\boldsymbol{X}$, y sea $\boldsymbol{y} = (x_i)_{X_i \in \boldsymbol{Y}}$ un valor de $\boldsymbol{Y} \subseteq \boldsymbol{X}$. Entonces, un modelo gráfico probabilístico para $\boldsymbol{X}$ es una factorización gráfica de la distribución de probabilidad generalizada conjunta $\rho(\boldsymbol{X} = \boldsymbol{x})$ (o simplemente $\rho(\boldsymbol{x})$). La representación del modelo se compone de una estructura gráfica E y un conjunto de distribuciones de probabilidad locales referidas a la estructura.

La estructura E para $\boldsymbol{X}$ es un grafo dirigido acíclico (DAG) donde los nodos van a ser las variables aleatorias. Dicha estructura describe las dependencias condicionales (Dawid, 1979) entre las variables en $\boldsymbol{X}$. $\boldsymbol{Pa}_i^E$ representa el conjunto de padres –variables de las que sale una flecha en E– de la variable X_i en el modelo gráfico probabilístico representado por la estructura E. La probabilidad de una determinada instanciación de las variables aleatorias es:

$$\begin{gathered}\rho(\boldsymbol{x}) = \rho(x_1, \ldots, x_n) \\ = \rho(x_1) \cdot \rho(x_2 \mid x_1) \cdot \ldots \cdot \rho(x_i \mid x_1, \ldots, x_{i-1}) \cdot \ldots \cdot \rho(x_n \mid x_1, \ldots, x_{n-1})\end{gathered}$$

y en virtud de la factorización que determina la estructura gráfica, podemos tener las distribuciones de probabilidad generalizadas locales, obteniendo la función de densidad:

$$\rho(\boldsymbol{x}) = \prod_{i=1}^{n} \rho(x_i \mid \boldsymbol{pa}_i^E). \tag{2.1}$$

Asumimos que las distribuciones de probabilidad generalizadas locales dependen de un conjunto finito de parámetros $\boldsymbol{\theta}_E = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$, por lo que la ecuación anterior

puede reescribirse como:

$$\rho(\boldsymbol{x} \mid \boldsymbol{\theta}_E) = \prod_{i=1}^{n} \rho(x_i \mid \boldsymbol{pa}_i^E, \boldsymbol{\theta}_i). \tag{2.2}$$

Después de haber definido los componentes del modelo gráfico probabilístico, podemos representarlo como un modelo compuesto por el par $M = (E, \boldsymbol{\theta}_E)$. En el caso particular de que cada variable $X_i \in \boldsymbol{X}$ sea discreta, llamaremos a dicho modelo *red Bayesiana*.

Al ser las variables discretas, X_i toma r_i posibles valores $x_i^1, \ldots, x_i^{r_i}$. Si denotamos por q_i el número de posibles instanciaciones diferentes de los padres de la variable X_i, esto es, $q_i = \prod_{X_g \in \boldsymbol{Pa}_i^E} r_g$ y como $\boldsymbol{pa}_i^{j,E}$ a la j-ésima instanciación, la distribución local para una variable concreta dada una determinada instanciación de sus padres será:

$$p({x_i}^k \mid \boldsymbol{pa}_i^{j,E}, \boldsymbol{\theta}_i) = \theta_{x_i^k \mid \boldsymbol{pa}_i^j} \equiv \theta_{ijk} \tag{2.3}$$

donde $\boldsymbol{pa}_i^{1,E}, \ldots, \boldsymbol{pa}_i^{q_i,E}$ denota los valores de $\boldsymbol{Pa}_i^E$, que es el conjunto de padres de la variable X_i en la estructura E. Los parámetros locales vienen representados por $\boldsymbol{\theta}_i = (((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i})$.

En otras palabras, los parámetros θ_{ijk} representan la probabilidad condicional de la variable X_i para su k-ésimo valor, cuando sus padres toman su j-ésima combinación de valores.

2.3 Aprendizaje de redes Bayesianas

Obtener una red Bayesiana a partir de un conjunto de datos es un proceso de aprendizaje que se divide en dos etapas: el aprendizaje estructural y el aprendizaje paramétrico. La primera de ellas, consiste en obtener la estructura de la red Bayesiana, es decir, las relaciones de dependencia e independencia condicional entre las variables involucradas. La segunda etapa, tiene como finalidad obtener las probabilidades a priori y condicionales requeridas a partir de una estructura dada.

Por tanto, para construir una red Bayesiana necesitamos especificar:

1.- Un grafo dirigido acíclico que refleja el conjunto de dependencias condicionales entre las variables. Este grafo es el objetivo del aprendizaje estructural.

2.- Las probabilidades a priori de todos los nodos raíz (cuyo conjunto de padres es vacío), esto es $p({x_i}^k \mid \emptyset, \boldsymbol{\theta}_i)$ o (θ_{i-k}), y el conjunto de las probabilidades condicionales para el resto de nodos, dadas todas las posibles combinaciones de sus predecesores directos, $p({x_i}^k \mid \boldsymbol{pa}_i^{j,E}, \boldsymbol{\theta}_i)$ o (θ_{ijk}), lo que llamamos aprendizaje paramétrico.

2.3.1 Aprendizaje estructural

Los modelos gráficos proporcionan una metodología sólida para manejar la incertidumbre en dominios complejos, ya que representan explícitamente de forma gráfica todas las dependencias condicionales entre variables. El problema de encontrar la estructura exacta es inviable, sobre todo si el número de variables es alto. Normalmente se realizan restricciones en el tipo de estructura a fin de reducir el espacio de búsqueda. Necesitaremos un algoritmo que genere estructuras más o menos óptimas y una función de evaluación que nos diga, para cada estructura generada, lo cercana que está de representar al conjunto de datos observados.

Existen dos metodologías para la obtención de la estructura de red, una basada en métodos de *score + search*, que utiliza un criterio para medir cómo la estructura de red se ajusta a los datos y un método de búsqueda para encontrar el modelo con mejor ajuste y otra metodología que mide el grado de relación de dependencia de los datos utilizando test de hipótesis estadísticos.

Un ejemplo de algoritmo de *score + search* para el aprendizaje de redes Bayesianas es el algoritmo $K2$ desarrollado por Cooper y Herskovits (Cooper y Herskovits, 1992). Este algoritmo define previamente un orden total entre las variables, y asume que todas las estructuras son a priori igualmente probables. Comienza asumiendo que cada nodo no tiene ningún nodo padre para, a continuación, en cada paso añadir aquel nodo padre cuya inclusión produce un mayor incremento de la probabilidad de la estructura resultante. El proceso termina cuando la adición de cualquier padre simple no incrementa la probabilidad. Otros ejemplos de scores utilizados son *Bayesian Information Criterion* (BIC) (Schwarz, 1978) o *Bayesian Dirichlet equivalence* (BDe) (Heckerman y col., 1995).

Como ejemplo de algoritmo que mide el grado de relación de dependencia entre las variables utilizando test de hipótesis está el algoritmo PC (Spirtes y col., 1991). Este algoritmo comienza con un grafo completo no dirigido y va eliminando arcos basándose en los resultados obtenidos al realizar test de independencia. Cuando no puede eliminar más arcos, completa el proceso dando una dirección a cada arco del grafo.

2.3.2 Aprendizaje paramétrico

El aprendizaje paramétrico consiste en la estimación de los parámetros numéricos del modelo gráfico probabilístico. Partimos de una estructura ya conocida y de un conjunto de datos asociados. En el caso de la red Bayesiana, estos parámetros son fácilmente extraíbles del conjunto de datos.

La figura 2.1 contiene un ejemplo de factorización de una red Bayesiana concreta con $\boldsymbol{X} = (X_1, X_2, X_3, X_4)$ y $r_2 = 3$, $r_i = 2$, para $i = 1, 3, 4$, siendo r_i el número de posibles valores que puede tomar la variable X_i.

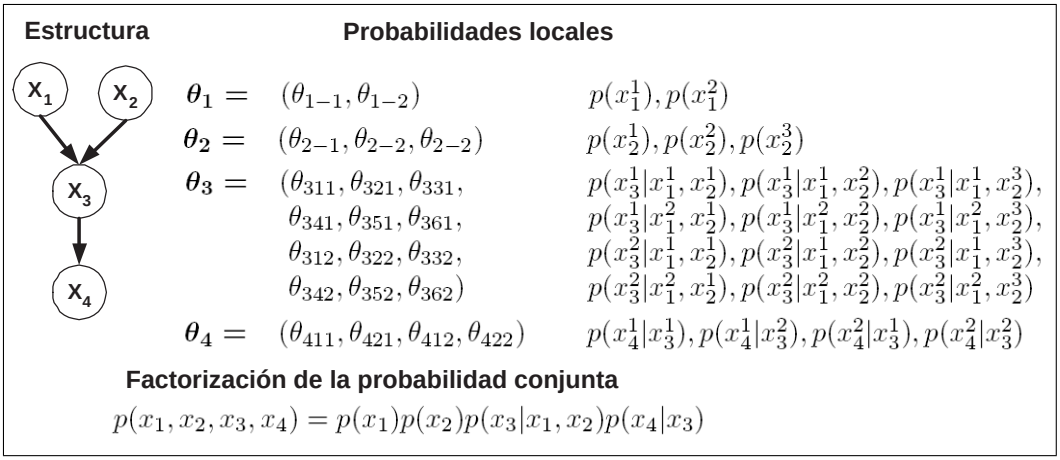

Figura 2.1: Estructura, probabilidades locales y función de factorización para una red Bayesiana con cuatro variables (X_1, X_3 y X_4 toman dos valores posibles, y X_2 con tres valores posibles).

2.4 Simulación de redes Bayesianas

La estructura de la red Bayesiana representa las dependencias probabilísticas entre las variables. Mediante la simulación de estas redes se pretende muestrear nuevos individuos teniendo en cuenta dichas dependencias probabilísticas. Utilizaremos la simulación de redes Bayesianas para generar nuevos individuos para la próxima población sobre la base de la estructura adquirida anteriormente. Este método crea una base de datos con las relaciones probabilísticas entre las diferentes variables. En el caso de las redes Bayesianas, se parte de una red donde el valor de las variables del sistema es desconocido y se utiliza para asignar aleatoriamente un valor a cada nodo. El valor de un nodo padre determina, en función de la probabilidad condicionada, el posible valor que puedan llegar a tener sus hijos.

Se han desarrollado multiples métodos de simulación de redes Bayesianas, como el *likelihood weighting method* desarrollado de manera independiente en (Fung y Chang, 1990) y (Shachter y Peot, 1990), y analizado posteriormente en (Shwe y Cooper, 1991), el muestreo hacia delante y hacia atrás (Fung y del Favero, 1994), el muestreo de Markov propuesto por Pearl (Pearl, 1987), o el método de muestreo sistemático (Bouckaert, 1994). En (Bouckaert y col., 1996) podemos encontrar una interesante comparación de los métodos anteriores aplicados a diferentes modelos de red Bayesiana utilizando, como criterios de comparación, el tiempo medio de ejecución del algoritmo y el error medio de propagación.

El método de simulación más utilizado para variables discretas es el denominado Probabilistic Logic Sampling (PLS) propuesto por Henrion (Henrion, 1988). En este método, la instanciación de cada variable se hace siguiendo un camino hacia delante,

es decir, una variable no se instancia hasta que no han sido instanciados todos sus padres. Esto requiere un ordenamiento previo de todas las variables de padres a hijos. Cualquier orden de las variables que satisfaga esa propiedad se conoce como orden ancestral. Denotamos por $\boldsymbol{\pi} = (\pi(1), \ldots, \pi(n))$ a un orden ancestral compatible con la estructura que queremos simular. Las variables son instanciadas de padres a hijos. Para cualquier red Bayesiana existe siempre al menos un orden ancestral, ya que los ciclos no están permitidos en las redes Bayesianas. Comenzamos utilizando un número aleatorio para instanciar cada variable raíz de la red, teniendo en cuenta su probabilidad a priori si la hubiera, y a continuación se simula un valor para cada variable del resto de la red, teniendo en cuenta la distribución de probabilidad $p(x_{\pi(i)} \mid \boldsymbol{pa}_i)$ donde $\boldsymbol{pa}_i$ representa los valores de los padres de la variable X_i.

Recientemente se ha desarrollado el uso de algoritmos BP(*Belief Propagation*) (Pearl, 1988; Nilsson, 1998; Yanover y Weiss, 2004) para la inferencia aproximada a partir de redes Bayesianas. Estos métodos se han utilizado en algunos trabajos con EDAs (Mendiburu y col., 2007; Soto, 2003) como una alternativa al PLS.

2.5 Redes Gaussianas

Una red Gaussiana es un modelo gráfico probabilístico en el que la estructura E es un grafo dirigido acíclico y todas las variables aleatorias que conforman los nodos son continuas. En este caso, la función de densidad local es un modelo de regresión normal (Whittaker, 1990).

$\boldsymbol{x} = (x_1, \ldots, x_n)$ representa a un individuo en el dominio continuo con valores en $\mathbb{R}^n$. La función de densidad local para la i-ésima variable X_i se puede calcular como un modelo de regresión lineal

$$f(x_i \mid \boldsymbol{pa}_i^E, \boldsymbol{\theta}_i) \equiv \mathcal{N}(x_i; m_i + \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_j - m_j), v_i) \tag{2.4}$$

donde $\mathcal{N}(x_i; \mu_i, \sigma_i^2)$ es una distribución normal univariada con media μ_i y varianza $v_i = \sigma_i^2$. Los parámetros locales vienen dados por $\boldsymbol{\theta}_i = (m_i, \boldsymbol{b}_i, v_i)$, donde m_i es la media incondicional de X_i, v_i es la varianza incondicional de X_i dados sus padres $\boldsymbol{Pa}_i$, $\boldsymbol{b}_i = (b_{1i}, \ldots, b_{i-1i})^t$ es un vector columna, y b_{ji} es el coeficiente lineal que mide la fuerza de la relación entre el nodo X_j y el nodo X_i.

Un modelo gráfico probabilístico construido a partir de estas funciones de densidad locales es lo que conocemos como *red Gaussiana* (Shachter y Kenley, 1989). La figura 2.2 representa un ejemplo de red Gaussiana en un espacio 4–dimensional.

Las redes Gaussianas y las densidades normales multivariantes están estrechamente relacionadas. Se considera que la función de densidad conjunta de una variable continua $\boldsymbol{X}$ es una distribución normal multivariante si:

$$f(\boldsymbol{x}) \equiv \boldsymbol{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) \equiv (2\pi)^{-\frac{n}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^t \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})} \tag{2.5}$$

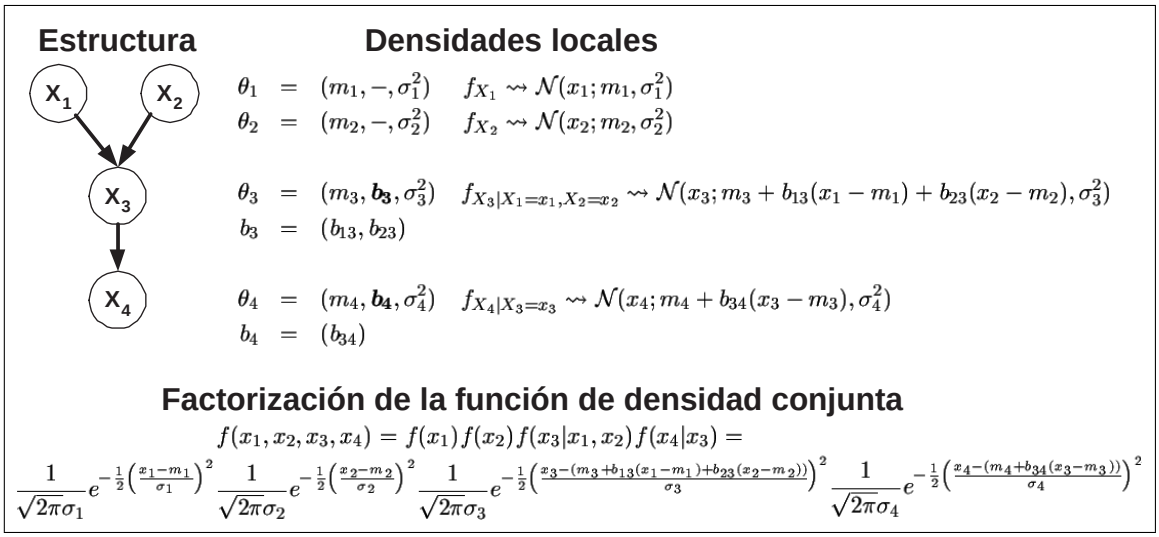

Figura 2.2: Estructura, densidades locales y resultados de factorización para una red Gaussiana con 4 variables.

donde $\boldsymbol{\mu}$ es el vector de medias, Σ es la matriz de varianzas-covarianzas $n \times n$, y $|\Sigma|$ es el determinante de Σ. La inversa de esta matriz, $W = \Sigma^{-1}$, cuyos elementos los denotamos por w_{ij}, es lo que llamamos matriz de precisión. Esta densidad podemos escribirla como un producto de n densidades condicionales:

$$f(\boldsymbol{x}) = \prod_{i=1}^{n} f(x_i \mid x_1, \ldots, x_{i-1}) = \prod_{i=1}^{n} \mathcal{N}(x_i; \mu_i + \sum_{j=1}^{i-1} b_{ji}(x_j - \mu_j), v_i) \tag{2.6}$$

donde μ_i es la media incondicional de X_i, v_i es la varianza de X_i dados $X_1, \ldots, X_{i-1}$, y b_{ji} es el coeficiente lineal que refleja la fuerza de relación entre las variables X_j y X_i (de Groot, 1970). Cuando b_{ji} es distinto de 0, determina la existencia de un arco entre X_j y X_i con $j < i$, de manera que si no hay un arco que va de X_j a X_i, $b_{ji} = 0$.

A partir de este concepto, podemos generar una densidad normal multivariante a partir de una red Gaussiana. Teniendo en cuenta que $m_i = \mu_i$ for all $i = 1, \ldots, n$, (Shachter y Kenley, 1989) describe el proceso general para construir la matriz de precisión W de la distribución normal representada por la red Gaussiana a partir de sus v_i and $\{b_{ji} \mid j < i\}$. Esta transformación se puede realizar utilizando la siguiente formula recursiva:

$$W(i+1) = \begin{cases} \frac{1}{v_1} & \text{si } i = 0 \\ \begin{pmatrix} W(i) + \frac{\boldsymbol{b}_{i+1}\boldsymbol{b}_{i+1}^t}{v_{i+1}}, & -\frac{\boldsymbol{b}_{i+1}}{v_{i+1}} \\ -\frac{\boldsymbol{b}_{i+1}^t}{v_{i+1}}, & \frac{1}{v_{i+1}} \end{pmatrix} & \text{si } i > 0 \end{cases} \tag{2.7}$$

Por ejemplo, aplicando el procedimiento anteriormente descrito al ejemplo de la

figura 2.2 donde $X_1 \equiv \mathcal{N}(x_1; m_1, v_1)$, $X_2 \equiv \mathcal{N}(x_2; m_2, v_2)$, $X_3 \equiv \mathcal{N}(x_3; m_3 + b_{13}(x_1 - m_1) + b_{23}(x_2 - m_2), v_3)$ y $X_4 \equiv \mathcal{N}(x_4; m_4 + b_{34}(x_3 - m_3), v_4)$, obtenemos la siguiente matriz de precisión W:

$$W = \begin{pmatrix} \frac{1}{v_1} + \frac{b_{13}^2}{v_3}, & \frac{b_{13}b_{23}}{v_3}, & -\frac{b_{13}}{v_3}, & 0 \\ \frac{b_{23}b_{13}}{v_3}, & \frac{1}{v_2} + \frac{b_{23}^2}{v_2}, & -\frac{b_{23}}{v_3}, & 0 \\ -\frac{b_{13}}{v_3}, & -\frac{b_{23}}{v_3}, & \frac{1}{v_3} + \frac{b_{34}^2}{v_4}, & -\frac{b_{34}}{v_4} \\ 0, & 0, & -\frac{b_{34}}{v_4}, & \frac{1}{v_4} \end{pmatrix} \tag{2.8}$$

2.6 Aprendizaje de redes Gaussianas

Igual que en caso de las redes Bayesianas, el aprendizaje de redes Gaussianas requiere de un aprendizaje estructural para obtener la estructura de la red y un aprendizaje paramétrico para estimar los parámetros numéricos.

2.6.1 Aprendizaje estructural

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 (Dempster, 1972). La idea es simplificar así la función de densidad normal n–dimensional conjunta. Esto es equivalente a realizar test de independencia condicional entre los elementos correspondientes de $\boldsymbol{X}$ (Wermuth, 1976) o a comprobar si el arco que conecta los vértices X_i y X_j en el grafo de independencia condicional puede ser eliminado (Speed y Kiiveri, 1986). Este método se conoce como test de exclusión de arcos. También existen métodos de búsqueda basados en métricas como los comentados para el caso de redes Bayesianas, los cuales también son validos para las redes Gaussianas aunque utilizando métricas diferentes a la hora de examinar la red. La función de verosimilitud $L(D \mid E, \boldsymbol{\theta})$ del conjunto de datos $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, dado el modelo de red Gaussiana $M = (E, \boldsymbol{\theta})$, es:

$$L(D \mid E, \boldsymbol{\theta}) = \prod_{r=1}^{N} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi v_i}} e^{-\frac{1}{2v_i}(x_{ir} - mi - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j))^2} \tag{2.9}$$

La estimación máximo verosimil para $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$, a saber $\widehat{\boldsymbol{\theta}} = (\widehat{\boldsymbol{\theta}}_1, \ldots, \widehat{\boldsymbol{\theta}}_n)$, se obtiene maximizando $L(D \mid E, \boldsymbol{\theta})$ o alternativamente, maximizando la expresión $\ln L(D \mid E, \boldsymbol{\theta})$. Tomando logaritmos de la expresión 2.9, obtenemos:

$$\begin{aligned} &\ln L(D \mid E, \boldsymbol{\theta}) \\ &= \sum_{r=1}^{N} \sum_{i=1}^{n} [-\ln(\sqrt{2\pi v_i}) - \frac{1}{2v_i}(x_{ir} - mi - \sum_{x_j \in \boldsymbol{pa}_i} b_{ji}(x_{jr} - m_j))^2] \end{aligned} \tag{2.10}$$

2.6.2 Aprendizaje paramétrico

A la hora de estimar los parámetros numéricos de una red Gaussiana, podemos obtener el estimador máximo verosímil $\widehat{\boldsymbol{\theta}} = (\widehat{\boldsymbol{\theta}}_1, \ldots, \widehat{\boldsymbol{\theta}}_n)$ del vector de parámetros de la red maximizando la expresión 2.10 y resolviendo el siguiente sistema de ecuaciones:

$$\begin{cases} \frac{\partial}{\partial m_i} \ln L(D \mid E, \boldsymbol{\theta}) = 0 \quad i = 1, \ldots, n \\ \\ \frac{\partial}{\partial v_i} \ln L(D \mid E, \boldsymbol{\theta}) = 0 \quad i = 1, \ldots, n \\ \\ \frac{\partial}{\partial b_{ji}} \ln L(D \mid E, \boldsymbol{\theta}) = 0 \quad j = 1, \ldots, i-1 \text{ y } X_j \in \boldsymbol{Pa}_i \end{cases} \tag{2.11}$$

obtenemos las 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_i (Larrañaga y Lozano, 2001).

2.7 Simulación de redes Gaussianas

Ripley (Ripley, 1987) propone dos aproximaciones generales al muestreo de distribuciones normales multivariadas, que pueden ser obtenidas a partir de la red Gaussiana. La primera aproximación se basa en la descomposición de Cholesky de la matriz de covarianzas. La segunda, conocida como método de condicionamiento, genera instancias de $\boldsymbol{X}$ calculando X_1, luego X_2 condicionado a X_1, etc, lo que tiene cierta similitud con el algoritmo PLS utilizado en las redes Bayesianas. La simulación de una distribución normal univariada se puede llevar a cabo por medio de un método simple basado en la suma de 12 variables uniformes independientes (Box y Muller, 1958).

Capítulo 3

Algoritmos de Estimación de Distribuciones

La computación evolutiva es una rama de la Inteligencia Artificial (IA) que se ocupa, entre otras cosas, de resolver problemas de optimización combinatoria. Se trata de instancias de problemas complejos en general, que se resuelven explorando el espacio de soluciones, generalmente grande, para estas instancias. Los algoritmos de optimización combinatoria logran esto reduciendo el tamaño efectivo del espacio, y explorando el espacio de búsqueda eficientemente.

Los *Algoritmos de Estimación de Distribuciones* (EDAs) son algoritmos basados en estrategias de búsqueda heurística estocástica dentro de la computación evolutiva. La principal característica de estos algoritmos consiste en crear un conjunto de soluciones de un problema particular (una población de individuos) y hacerla evolucionar de manera que en cada iteración se obtenga un nuevo conjunto de soluciones cada vez más próximas a la solución optima. Hasta aquí, la idea coincide con la del método que siguen los algoritmos genéticos (GAs). La diferencia entre ambos métodos está en el proceso de evolución hacia poblaciones más prometedoras. Mientras que los GAs realizan esta evolución mediante operaciones de cruce y mutación, los EDAs lo sustituyen por la estimación y posterior muestreo de una distribución de probabilidad conjunta de las variables a partir de los mejores individuos (soluciones) de la generación anterior. Es decir, estos algoritmos representan, mediante un modelo gráfico probabilístico, las relaciones entre las variables. Este conjunto de algoritmos ha sido objeto de gran atención por parte de la comunidad científica en los últimos años (Larrañaga y Lozano, 2001; Lozano y col., 2006; Mühlenbein y Paaß, 1996; Pelikan y col., 2006).

En este apartado explicaremos el funcionamiento general de los EDAs y realizaremos una revisión de los diferentes algoritmos existentes, prestando especial interés al número de dependencias entre variables que tienen en cuenta.

3.1 Esquema general de un algoritmo de estimación de distribuciones

La idea principal en los Algoritmos de Estimación de las Distribuciones (EDAs) (Larrañaga y Lozano, 2001; Mühlenbein y Paaß, 1996; Pelikan y col., 1999) es mantener una población de individuos (o un conjunto de soluciones del problema particular) y hacerlos evolucionar para obtener en cada iteración una población de individuos mejor. Cada individuo es un vector de valores considerados como una instanciación de las variables estadísticas. En los EDAs se genera una nueva población de individuos a partir del modelo gráfico probabilístico. Este modelo gráfico probabilístico se obtiene en base a los datos de los individuos seleccionados de la generación anterior, y las interrelaciones entre las diferentes variables que representan a cada individuo se expresan explícitamente por la distribución de probabilidad asociada a los individuos seleccionados en cada iteración.

3.1.1 Notación

Comenzaremos por introducir la notación que utilizaremos en este capítulo, antes de explicar con más detalle los diferentes pasos que siguen los EDAs.

Sea $\boldsymbol{X} = (X_1, \ldots, X_n)$ una variable aleatoria n–dimensional, donde $\boldsymbol{x} = (x_1, \ldots, x_n)$ representa una posible instanciación de cada uno de los posibles individuos. La función objetivo o función de idoneidad, $F(\boldsymbol{x})$, mide la bondad de cada individuo. La probabilidad de $\boldsymbol{X}$ la representaremos por $p(\boldsymbol{X} = \boldsymbol{x})$ –o simplemente $p(\boldsymbol{x})$. La probabilidad condicional de que la variable X_i tome el valor x_i dado el valor x_j de la variable X_j, podemos representarla como $p(X_i = x_i | X_j = x_j)$ –o simplemente por $p(x_i|x_j)$. Llamaremos D_l a la l-ésima población de R individuos que tiene que evolucionar hacia la población $(l+1)$-ésima. Para el proceso de aprendizaje, los EDAs seleccionan $S < R$ individuos de D_l y el resto sencillamente se ignoran. Llamaremos D_l^S al subconjunto de D_l que utilizaremos para el aprendizaje.

3.1.2 Descripción de los principales pasos de los EDAs

Por lo general, un EDA realiza los siguientes pasos:

1. Generación de la primera población. En primer lugar, se genera la primera población D_0 de R individuos. Para ello, normalmente se asume una distribución uniforme sobre cada variable. A continuación se evalúa cada individuo, utilizando para ello la función objetivo que guiará todo el proceso.

2. Selección de individuos. Se seleccionan un número S $(S < R)$ de individuos de D_l. Normalmente, el criterio que se utiliza para esta selección se basa en el valor obtenido por el individuo en la función objetivo, de manera que normalmente se selecciona a los mejores. Estos individuos forman la población seleccionada D_l^S.

EDAs

$D_0 \leftarrow$ Generar R individuos (población inicial) al azar

Repetir Para cada $l = 1, 2, \ldots$ hasta satisfacer un criterio de parada

$D_{l-1}^S \leftarrow$ Seleccionar $S < R$ individuos de D_{l-1} de acuerdo a un método de selección

$p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_{l-1}^S) \leftarrow$ Estimar la distribución de probabilidad de que un individuo se encuentre entre los seleccionados

$D_l \leftarrow$ Crear R nuevos individuos (la nueva población) a partir de $p_l(\boldsymbol{x})$

Figura 3.1: Pseudocódigo del enfoque EDAs.

3. Aprendizaje del modelo. A continuación, se induce el modelo probabilístico n-dimensional $p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_l^S)$ que representa las interdependencias entre las n variables. Este modelo se crea como un modelo gráfico probabilístico conteniendo a las variables $X_1, X_2, \ldots, X_n$, donde n es el tamaño de cada individuo. En las siguientes secciones 3.2 y 3.3, comentamos con más detalles como se realiza este proceso, que será diferente dependiendo del algoritmo utilizado.

4. Simulación de nuevos individuos. La nueva población D_{l+1} formada por los R nuevos individuos se obtiene simulando la distribución de probabilidad aprendida en el paso anterior. En el capítulo anterior se comentó como se puede realizar la simulación de redes Bayesianas y Gaussianas.

5. Criterio de parada. Los pasos 2, 3 y 4 se repiten hasta que se cumpla un criterio de parada, por ejemplo alcanzar un número fijo de poblaciones o un número fijo de individuos diferentes, uniformidad en la población generada, o el hecho de haber llegado a la solución óptima (si esta se conoce).

La figura 3.1 muestra el pseudocódigo de los EDAs en problemas de optimización combinatorial. En la figura 3.2 podemos ver una ilustración gráfica de este proceso.

3.2 Algoritmos de estimación de distribuciones en entornos discretos

La parte más critica de los EDAs de cara a realizar la optimización, radica precisamente en realizar la estimación de la distribución de probabilidad conjunta (Gámez y col., 2007; Santana, 2005; Santana y col., 2006). La idea básica consiste en inducir modelos probabilísticos a partir de los mejores individuos de cada población. La

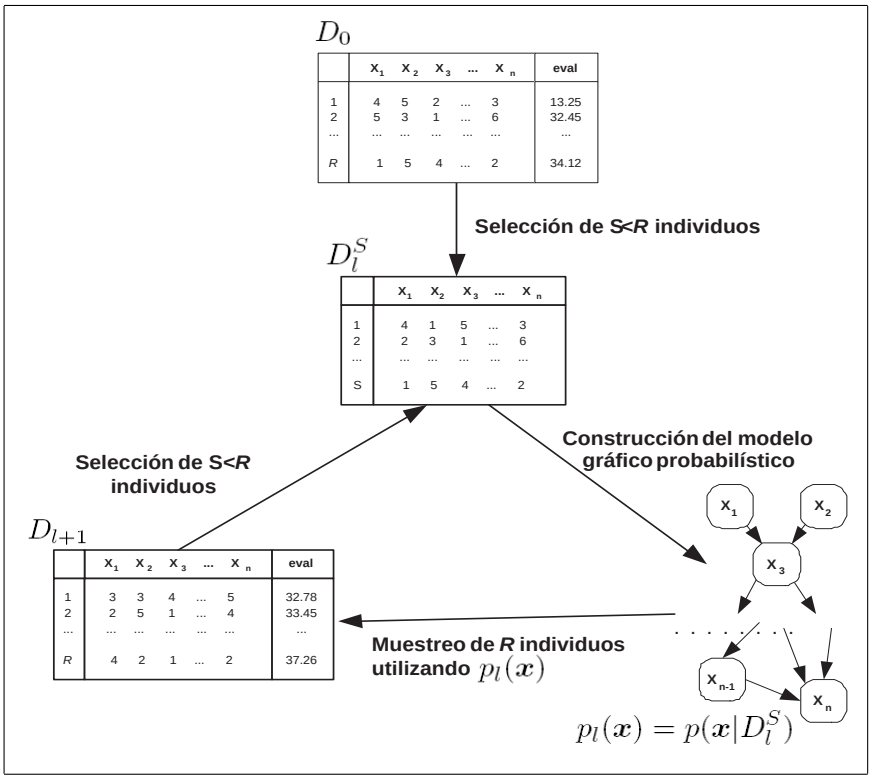

Figura 3.2: Ilustración del método EDA en el proceso de optimización.

clave está en encontrar el modelo probabilístico n-dimensional que mejor exprese las interdependencias entre las n-variables. Las relaciones entre la estructura del problema y las dependencias que aparecen en el conjunto seleccionado han sido analizadas por diferentes autores (Bengoetxea, 2003; Mühlenbein y Höns, 2005; Echegoyen y col., 2007; Hauschild y col., 2007; Echegoyen y col., 2008). Una vez que el modelo ha sido estimado, los individuos de la siguiente generación (las nuevas soluciones) se generan por medio de la simulación del modelo, y un subconjunto de estos, habitualmente los mejores, se utilizarán a su vez para crear el modelo de la siguiente generación. Este proceso se repite en cada generación.

En esta sección explicamos algunos ejemplos de EDAs que se pueden encontrar en la literatura. Todos los algoritmos y métodos se clasifican dependiendo del número máximo de dependencias entre las variables que se consideran (número máximo de padres que puede tener una variable X_i en el modelo gráfico probabilístico). Aunque no es nuestro objetivo realizar una revisión completa de estos algoritmos, que se puede encontrar con todo detalle en (Larrañaga y Lozano, 2001).

3.2.1 Independencia entre variables

Los métodos que pertenecen a esta categoría consideran que todas las variables son independientes y asumen que la distribución de probabilidad conjunta n–dimensional se puede factorizar como un producto de n distribuciones de probabilidad univariantes e independientes. Es decir,

$$p_l(\boldsymbol{x}) = \prod_{i=1}^{n} p_l(x_i) \tag{3.1}$$

Esta suposición no es exacta para muchos de los problemas en los que se aplican técnicas de optimización, donde siempre existen interdependencias de algún tipo entre las variables. Sin embargo, se ha reconocido (Mühlenbein y col., 1999; Santana y col., 2005) que no es necesario representar todas las dependencias para resolver de manera eficiente, un problema determinado, utilizando EDAs. Esta es una aproximación sencilla que puede conducir a un comportamiento aceptable de los EDAs para problemas en los que se puede asumir la independencia entre las variables, estando su uso motivado por el bajo costo computacional asociado.

Como ejemplos de algoritmos basados en esta asunción de independencia podemos mencionar los siguientes:

- El algoritmo UMDA (Univariate Marginal Distribution Algorithm) (Mühlenbein, 1998).
- El algoritmo BSC (Bit-Based Simulated Crossover) (Syswerda, 1993).
- El aprendizaje incremental basado en poblaciones–PBIL–(Population-Based Incremental Learning) (Baluja, 1994).
- El algoritmo genético compacto –cGA– (Harik y col., 1998).

A continuación describimos más detalladamente los algoritmos UMDA y BSC como ejemplos de esta propuesta.

UMDA –Univariate Marginal Distribution Algorithm

Este algoritmo fue propuesto por Mühlenbein (Mühlenbein, 1998). La distribución de probabilidad conjunta se factoriza como producto de distribuciones univariantes independientes. Esto es:

$$p_l(\boldsymbol{x}) = p(\boldsymbol{x} \mid D_{l-1}^S) = \prod_{i=1}^{n} p_l(x_i). \tag{3.2}$$

En este caso, cada distribución marginal univariante se estima a partir de las frecuencias marginales:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i \mid D_{l-1}^S)}{N} \tag{3.3}$$

donde

$$\delta_j(X_i = x_i \mid D_{l-1}^S) = \begin{cases} 1 & \text{si en el } j\text{-ésimo caso de } D_{l-1}^S, X_i = x_i \\ 0 & \text{en otro caso.} \end{cases} \tag{3.4}$$

BSC –Bit-Based Simulated Crossover

Este algoritmo (Syswerda, 1993) utiliza el valor de la función de evaluación, $F(\boldsymbol{x})$, de los individuos seleccionados para estimar la distribución marginal de cada uno:

$$p_l(x_i) = \frac{\sum_{\{\boldsymbol{x}|\delta_j(X_i=x_i|D_{l-1}^S)=1\}} F(\boldsymbol{x})}{\sum_{\{\boldsymbol{x}\in D_{l-1}^S\}} F(\boldsymbol{x})} \tag{3.5}$$

donde la función δ_j mantiene la misma expresión que en la ecuación 3.3.

En la ecuación 3.5, el numerador representa la suma de los valores obtenidos en la función de evaluación por los individuos con valor x_i en la variable X_i, y el denominador es la suma de los valores obtenidos en la función de evaluación por todos los individuos seleccionados.

3.2.2 Dependencias entre pares de variables

En este apartado incluimos aquellos algoritmos que consideran que la mayoría de las dependencias son entre pares de variables, de manera que consideran dependencias pero nunca entre más de dos variables. Mientras que en los algoritmos del apartado anterior simplemente era necesario realizar un aprendizaje paramétrico, ya que la estructura permanecía fija, en este caso se requiere un paso adicional que consiste en la inducción de la estructura que mejor represente el modelo probabilístico (aprendizaje estructural). La figura 3.3 muestra ejemplos de posibles estructuras gráficas donde se expresan estas dependencias entre pares de variables.

Como ejemplos de EDA de esta segunda categoría podemos mencionar los siguientes:

- El algoritmo voraz llamado MIMIC (Mutual Information Maximization for Input Clustering), propuesto por de Bonet y col. (de Bonet y col., 1997).
- El COMIT (Combining Optimizers with Mutual Information Trees) propuesto por Baluja y Davies (Baluja y Davies, 1997) combina EDAs con optimización local. La estimación de la distribución de probabilidad de los individuos seleccionados en cada generación se hace mediante una red Bayesiana con estructura de árbol, utilizando en el aprendizaje el algoritmo Maximum Weight Spanning Tree (MWST) propuesto Chow-Liu (Chow y Liu, 1968).
- El llamado BMDA (Bivariate Marginal Distribution Algorithm) propuesto por Pelikan y Mühlenbein (Pelikan y Mühlenbein, 1999). Este algoritmo utiliza una factorización de la distribución de probabilidad conjunta que sólo necesita estadísticos de segundo orden.

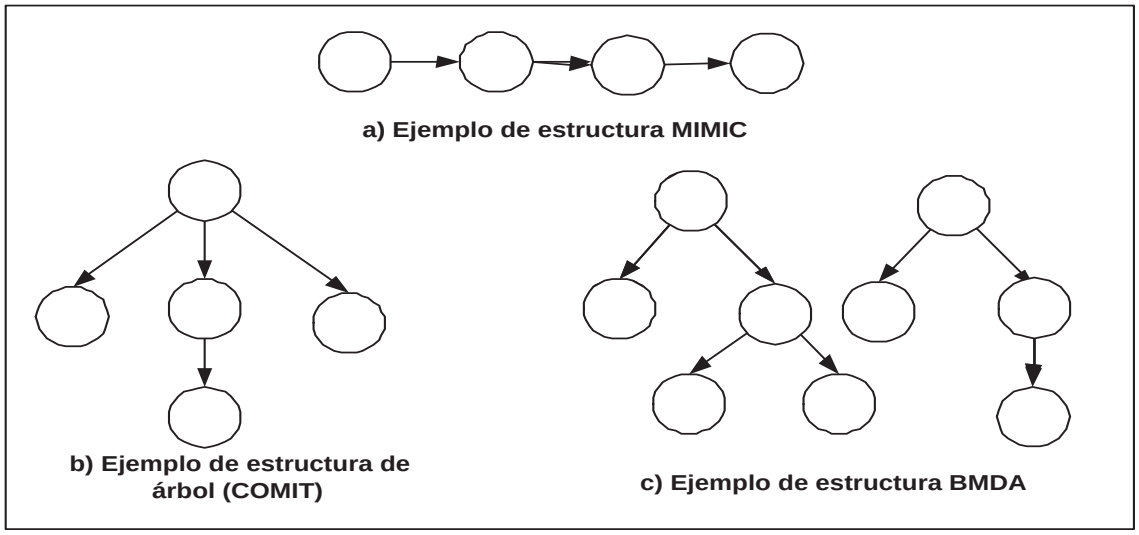

Figura 3.3: Ejemplos de representaciones gráficas de algunos EDAs con dependencias entre pares de variables (MIMIC, estructura de árbol de COMIT, BMDA).

MIMIC –Mutual Information Maximization for Input Clustering

Este algoritmo busca la mejor permutación entre las variables, en cada generación. El objetivo es encontrar la distribución de probabilidad $p_l^{\pi}(\boldsymbol{x})$, que se encuentra más cercana en divergencia de Kullback-Leibler a la distribución empírica del conjunto de individuos seleccionados.

$$p_l^{\pi}(\boldsymbol{x}) = p_l(x_{i_1} \mid x_{i_2}) \cdot p_l(x_{i_2} \mid x_{i_3}) \cdot \ldots \cdot p_l(x_{i_{n-1}} \mid x_{i_n}) \cdot p_l(x_{i_n}) \tag{3.6}$$

donde $\pi = (i_1, i_2, \ldots, i_n)$ representa una permutación de los índices $1, 2, \ldots, n$.

La divergencia de Kullback-Leibler entre dos distribuciones de probabilidad, $p_l(\boldsymbol{x})$ y $p_l^{\pi}(\boldsymbol{x})$, se puede expresar como:

$$H_l^{\pi}(\boldsymbol{x}) = h_l(X_{i_n}) + \sum_{j=1}^{n-1} h_l(X_{i_j} \mid X_{i_{j+1}}) \tag{3.7}$$

donde

$$h(X) = -\sum_x p(X = x) \log p(X = x) \tag{3.8}$$

denota la entropía de Shannon de la variable X, y

$$h(X \mid Y) = \sum_y h(X \mid Y = y) p(Y = y) \tag{3.9}$$

1.- Seleccionar $i_n = \arg\min_j \widehat{h}_l(X_j)$ *//variable con menor entropía*

2.- **for** $k = n-1, n-2, \ldots, 2, 1$

Escoger $i_k = \arg\min_j \widehat{h}_l(X_j \mid X_{i_{k+1}}), \quad j \neq i_{k+1}, \ldots, i_n$

3.- $p_l^{\pi}(\boldsymbol{x}) = p_l(x_{i_1} \mid x_{i_2}) \cdot p_l(x_{i_2} \mid x_{i_3}) \cdot \ldots \cdot p_l(x_{i_{n-1}} \mid x_{i_n}) \cdot p_l(x_{i_n})$

Figura 3.4: Método MIMIC para estimar la distribución de probabilidad conjunta.

donde

$$h(X \mid Y = y) = -\sum_x p(X = x \mid Y = y) \log p(X = x \mid Y = y) \tag{3.10}$$

denota la incertidumbre media de X dado Y. Se trata de encontrar la permutación π^* que minimice $H_l^{\pi}(\boldsymbol{x})$. Dado que buscar las $n!$ posibles permutaciones no es posible, MIMIC busca la mejor permutación seleccionando como variable X_{i_n} aquella con mejor entropía estimada (esta selección se realiza en D_l^S) y en cada paso, selecciona la variable con menor entropía condicional, con respecto a la variable seleccionada en el paso anterior.

La figura 3.4 muestra el pseudocódigo del algoritmo MIMIC para la estimación de la distribución de probabilidad conjunta en la generación l. $\widehat{h}_l(X)$ y $\widehat{h}_l(X \mid Y)$ representan la entropía empírica de X y de X dado Y respectivamente.

3.2.3 Múltiples dependencias

Por último tendríamos algoritmos que consideran múltiples dependencias entre variables. Este tipo de algoritmos proponen la construcción de un modelo gráfico probabilístico sin restricciones en el número de padres que cada variable pueda tener.

Como el número de dependencias entre variables es mayor que en los casos anteriores, la complejidad de la estructura probabilística, así como la tarea de encontrar la mejor estructura que represente el modelo es mucho mayor. Por tanto esta opción requiere de un proceso de aprendizaje más complejo.

Como ejemplos de algoritmos de esta categoría en el dominio discreto, podemos mencionar los siguientes:

- El algoritmo EcGA (Extended compact Genetic Algorithm) presentado por Harik (Harik y col., 1998). Este algoritmo divide a las variables en grupos, cada uno de los cuales se considera independiente del resto. El agrupamiento de las variables se lleva a cabo por medio de un algoritmo voraz hacia delante mediante el cual se obtiene una partición de las n variables. La idea es utilizar, en cada ge-

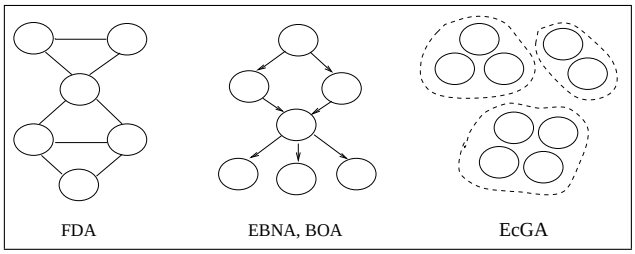

Figura 3.5: Ejemplos de representaciones gráficas de algunos EDAs con múltiples dependencias (FDA, EBNA, BOA y EcGA).

neración, un modelo de factorización de la distribución de probabilidad conjunta como producto de marginales de tamaño variable.

- El algoritmo FDA o algoritmo de distribución factorizada (Factorized Distribution Algorithm) se presenta en el trabajo de Mühlenbein y col. (Mühlenbein y col., 1999). Este algoritmo trata de optimizar funciones aditivamente descomponibles, para las cuales se obtiene una factorización de la distribución de probabilidad basada en residuales y separadores.

- El algoritmo EBNA (Estimation of Bayesian Networks Algorithm o algoritmo de estimación de redes Bayesianas) se presenta por Etxeberria y Larrañaga (Etxeberria y Larrañaga, 1999). En este algoritmo la distribución de probabilidad conjunta, que está codificada en una red Bayesiana, se aprende a partir de los individuos seleccionados en cada generación.

- Pelikan y col. (Pelikan y col., 1999) proponen un algoritmo llamado BOA (Bayesian Optimization Algorithm) en el que utilizan la métrica BDe (Heckerman y col., 1995), que devuelve el mismo valor si las redes reflejan el mismo conjunto de dependencias condicionales.

- Mühlenbein y Mahnig (Mühlenbein y Mahning, 1999) proponen el algoritmo LFDA (Learning Factorized Distribution Algorithm) que básicamente sigue la misma aproximación que el EBNA. La diferencia radica en que en LFDA la complejidad del modelo se controla por la métrica BIC en conjunción con una restricción acerca del máximo número de padres que cada variable puede llegar a tener.

La figura 3.5 muestra diversos modelos gráficos probabilísticos que se incluyen en esta categoría.

EBNA –Estimation of Bayesian Network Algorithm

Como ya se ha comentado anteriormente, EBNA es un EDA propuesto por Etxeberria y Larrañaga (Etxeberria y Larrañaga, 1999). Este algoritmo propone la construcción de un modelo gráfico probabilístico (una red Bayesiana) sin restringir el número de padres que puede tener una variable. Se comienza con un grafo acíclico dirigido, sin arcos. En este caso, la factorización de la distribución de probabilidad conjunta se obtiene a partir del producto de las n distribuciones uniformes de probabilidad marginales. Es decir, que la red Bayesiana inicial asigna la misma probabilidad a todos los puntos del espacio de búsqueda.

A la hora de aprender la estructura es preferible un algoritmo simple que obtenga una buena estructura en poco tiempo, aunque esta no sea la estructura optima, ya que este proceso se realiza en cada generación. Un algoritmo que cumple estas características es el algoritmo B (Buntine, 1991). Este algoritmo es un algoritmo voraz que comienza con una estructura sin arcos y en cada paso añade el arco que proporciona mayor mejora en la métrica utilizada. Este algoritmo termina cuando añadir cualquier arco no supone ninguna mejora. También podemos obtener buenas estructuras de una manera rápida utilizando estrategias de búsqueda local. Estas estrategias comienzan la búsqueda con el modelo obtenido en la generación anterior.

Dependiendo del método utilizado para el aprendizaje estructural de la red Bayesiana podemos encontrar diferentes versiones del algoritmo EBNA: De esta forma, EBNA_{K2+pen} y EBNA_{BIC} realizan la búsqueda de la estructura para detectar dependencias aplicando métodos de búsqueda y score, mientras que EBNA_{PC} utiliza test de hipótesis estadísticas a partir de las cuales estimar la estructura de la red.

La figura 3.6 muestra el pseudocódigo de los algoritmos EBNA_{PC}, EBNA_{K2+pen} y EBNA_{BIC}, donde M_l se refiere a la red Bayesiana obtenida en la l-ésima generación.

3.3 Algoritmos de estimación de distribuciones en entornos continuos

En esta sección se presentan algunos EDAs desarrollados para la optimización de problemas que utilizan variables continuas. Sin embargo, podemos encontrar gran cantidad de trabajos relacionados con el análisis y desarrollo de EDAs en entornos continuos (Bosman y Grahl, 2008; Bosman y Thierens, 2006; Larrañaga y col., 2000). El esquema general de funcionamiento es el mismo que el mostrado en la figura 3.1, teniendo en cuenta que en el dominio continuo, la función de densidad conjunta, se puede factorizar como el producto de n funciones de densidad condicionales. Se ha utilizado la misma notación que en la sección anterior, sustituyendo la distribución de probabilidad $p_l(\boldsymbol{x})$ por la función de densidad $f_l(\boldsymbol{x})$.

Esta sección se ha organizado de la misma manera que en el caso anterior para valores discretos, clasificando los algoritmos en diferentes categorías según el número

EBNA$_{PC}$, EBNA$_{K2+pen}$, EBNA$_{BIC}$

$M_0 \leftarrow (S_0, \boldsymbol{\theta}^0)$ con S_0 DAG sin arcos y θ^0 uniforme.
$p_0(x) = \prod_{i=1}^{n} p(x_i) = \prod_{i=1}^{n} \frac{1}{r_i}$

$D_0 \leftarrow$ Muestrear R individuos de M_0

For $l = 1, 2, \ldots$ hasta satisfacer un criterio de parada
- $D_{l-1}^S \leftarrow$ Seleccionar S individuos de D_{l-1}
- $S_l^* \leftarrow$ Encontrar la mejor estructura según un criterio:
  - test de independencia condicional (EBNA$_{PC}$)
  - Score Bayesiano penalizado + búsqueda (EBNA$_{K2+pen}$)
  - Máxima verosimilitud penalizada + búsqueda (EBNA$_{BIC}$)
- $\boldsymbol{\theta}^l \leftarrow$ Calcular $\{\theta_{ijk}^l = \frac{N_{ijk}^{l-1}+1}{N_{ij}^{l-1}+r_i}\}$ usando D_{l-1}^S como conjunto de datos
- $M_l \leftarrow (S_l^*, \boldsymbol{\theta}^l)$
- $D_l \leftarrow$ Muestrear R individuos de M_l utilizando PLS

Figura 3.6: Pseudocódigo de los algoritmos EBNA$_{PC}$, EBNA$_{K2+pen}$ y EBNA$_{BIC}$.

de dependencias entre variables que consideren.

3.3.1 Independencia entre variables

Se incluyen en esta categoría aquellos algoritmos que no tienen en cuenta ninguna dependencia entre las variables. En este caso, la función de densidad conjunta sigue una distribución normal n-dimensional que se factoriza como el producto de densidades normales unidimensionales independientes:

$$f_{\mathcal{N}}(\boldsymbol{x}; \mu, \sum) = \prod_{i=1}^{n} f_{\mathcal{N}}(x_i; \mu_i, \sigma_i^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}(\frac{x_i-\mu_i}{\sigma_i})^2}. \tag{3.11}$$

Como ejemplos de EDAs continuos de esta categoría están la versión continua del UMDA (UMDA$_c$) (Larrañaga y col., 2000), una versión continua del algoritmo de aprendizaje incremental basado en poblaciones (PBIL$_c$) (Sebag y Ducoulombier, 1998) y el algoritmo SHCLVND (Stochastic Hill-Climbing with Learning by Vectors of Normal Distributions) (Rudlof y Köppen, 1996). A continuación se describe el algoritmo UMDAc con más detalle.

UMDA$_c$

** Aprendizaje de la función de densidad conjunta **
for $l = 1, 2, \ldots$ hasta satisfacer un criterio de parada
  for $i = 1 \ldots n$
    (i) seleccionar mediante tests de hipótesis la función de densidad $f_l(x_i; \boldsymbol{\theta}_i^l)$ que mejor se adapta a D_{l-1}^{S,X_i}, la proyección de los individuos seleccionados sobre la variable $i - ésima$
    (ii) obtener la estimación de máxima verosimilitud para $\boldsymbol{\theta}_i^l = (\theta_i^{l,k_1}, \ldots, \theta_i^{l,k_i})$

El aprendizaje de la función de densidad conjunta en cada generación, se expresa como:
$f_l(\boldsymbol{x}; \boldsymbol{\theta}^l) = \prod_{i=1}^n f_l(x_i, \widehat{\boldsymbol{\theta}}_i^{\,l})$

Figura 3.7: Pseudocódigo para estimar la función de densidad conjunta utilizado en UMDA$_c$.

UMDA$_c$

La versión para dominios continuos del algoritmo UMDA (Univariate Marginal Distribution Algorithm) (UMDA$_c$) fue desarrollada por Larrañaga y colaboradores (Larrañaga y col., 2000). En este caso, la factorización de la función de densidad conjunta viene dada por

$$f_l(\boldsymbol{x}; \boldsymbol{\theta}^l) = \prod_{i=1}^{n} f_l(x_i, \boldsymbol{\theta}_i^l). \tag{3.12}$$

UMDA$_c$ identifican mediante tests de hipótesis las densidades unidimensionales del modelo. Una vez identificadas las densidades, la estimación de los parámetros se lleva a cabo por medio de sus estimaciones de máxima verosimilitud, tal y como se explicó en la sección 2.6. En el caso de que las densidades unidimensionales sigan distribuciones normales, en cada generación, tenemos que estimar la media μ_i^l y la desviación estándar σ_i^l para cada variable. Sabemos que sus respectivas estimaciones de máxima verosimilitud son:

$$\widehat{\mu_i}^{\,l} = \overline{X_i}^{\,l} = \frac{1}{N}\sum_{r=1}^{N} x_{i,r}^l; \qquad \widehat{\sigma_i}^{\,l} = \sqrt{\frac{1}{N}\sum_{r=1}^{N}\left(x_{i,r}^l - \overline{X_i}^{\,l}\right)^2} \tag{3.13}$$

La figura 3.7 muestra el pseudocódigo que utiliza el UMDA$_c$ para aprender la función de densidad conjunta.

3.3.2 Dependencias entre pares de variables

MIMIC$_c^G$

El algoritmo MIMIC$_c^G$ es una adaptación del algoritmo MIMIC para dominios continuos, presentada por Larrañaga y colaboradores (Larrañaga y col., 2000). En este caso se asume que el modelo probabilístico subyacente, para cada par de variables, es una función de densidad Gaussiana bivariante.

La idea consiste en usar una única densidad univariada y $n-1$ densidades condicionales bivariadas para definir la función de densidad conjunta, tratando de ajustarla lo máximo posible a los datos empíricos. Para evaluar la red, se basan en el siguiente resultado (Whittaker, 1990):

Si $\boldsymbol{X} \equiv \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \sum)$ en una función de densidad normal n–dimensional, entonces su entropía viene dada por la expresión:

$$h(\boldsymbol{X}) = \frac{1}{2} n(1 + \log 2\pi) + \frac{1}{2} \log \mid \sum \mid . \tag{3.14}$$

Por tanto, la expresión para la densidad univariada será:

$$h(X) = \frac{1}{2}(1 + \log 2\pi) + \log \sigma_X \tag{3.15}$$

y la de las bivariadas:

$$h(X \mid Y) = \frac{1}{2} \left[(1 + \log 2\pi) + \log \left(\frac{\sigma_X^2 \sigma_Y^2 - \sigma_{XY}^2}{\sigma_Y^2} \right) \right] \tag{3.16}$$

donde σ_X^2 es la varianza de la variable unidimensional X y σ_{XY} es la covarianza entre las variables X e Y.

El aprendizaje de la estructura que se realiza en este algoritmo se muestra en la figura 3.8. En un primer paso se elige la variable con menor varianza de todas (que será la asociada a la densidad univariada) y en un segundo paso se selecciona la variable que será el próximo hijo. Para ello se prueba con todas las variables no escogidas hasta el momento y se elige aquella cuyo valor de $\frac{\hat{\sigma}_X^2 \hat{\sigma}_Y^2 - \hat{\sigma}_{XY}^2}{\hat{\sigma}_Y^2}$ sea menor, para así bajar la entropía, que es la expresión que se usa como distancia con respecto a la información que aparece en la población en curso.

3.3.3 Múltiples dependencias

Los algoritmos que se incluyen esta sección corresponden a EDAs en el dominio continuo, en los que no hay ninguna restricción sobre el número de dependencias entre variables a tener en cuenta en el aprendizaje del modelo. Esto puede hacerse induciendo un modelo de distribución normal multivariante no restrictivo, como en el caso del algoritmo EMNA$_{global}$ (Estimation of Multivariate Normal Algorithm) (Larrañaga y

MIMIC$_c^G$

Elige $i_n = \arg\min_j \ \hat{\sigma}^2_{X_j}$

for $k = n-1, n-2, \ldots, 1$

Elige $i_k = \arg\min_j \ \hat{\sigma}^2_{X_j} - \frac{\hat{\sigma}^2_{X_j X_{i_{k+1}}}}{\hat{\sigma}^2_{X_{i_{k+1}}}}$

$j \neq i_{k+1}, \ldots, i_n$

Figura 3.8: Adaptación del método MIMIC a funciones de densidad Gaussianas multivariantes.

EMNA$_{global}$

$D_0 \leftarrow$ Generar R individuos al azar (la población inicial)

for $l = 1, 2, \ldots$ hasta alcanzar un criterio de parada

$D^S_{l-1} \leftarrow$ Seleccionar $S < R$ individuos de D_{l-1} de acuerdo a un método de selección

$f_l(\boldsymbol{x}) = f(\boldsymbol{x} \mid D^S_{l-1}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_l, \Sigma_l) \leftarrow$ Calcular la función de densidad normal multivariante, a partir de los individuos seleccionados

$D_l \leftarrow$ Muestrear R individuos (la nueva población) a partir de $f_l(\boldsymbol{x})$

Figura 3.9: Pseudocódigo del algoritmo EMNA$_{global}$.

Lozano, 2001) o aprendiendo una red Gaussiana como se hace en los algoritmos EGNA (Estimation of Gaussian Network Algorithm) (Larrañaga y col., 2000).

EMNA$_{global}$

En este método, la factorización de los individuos seleccionados en cada generación se obtiene mediante una función de densidad normal multivariante. La figura 3.9 muestra el pseudocódigo del algoritmo EMNA$_{global}$. En este algoritmo, en cada generación se calculan los estimadores de máxima verosimilitud para el vector de medias $\boldsymbol{\mu}_l = (\mu_{1,l}, \ldots, \mu_{n,l})$, y para la matriz de varianzas-covarianzas $\Sigma_l = \sigma^2_{ij,l}$ con $i, j = 1, \ldots, n$. Por lo tanto, en cada generación es necesario calcular $2n + \binom{n-1}{2}$ parámetros: n medias, n varianzas y $\binom{n-1}{2}$ covarianzas. Estos estimadores de

máxima verosimilitud se calculan de la siguiente manera:

$$\hat{\mu}_{i,l} = \frac{1}{S}\sum_{r=1}^{S} x_{i,r}^{l} \quad i = 1, \ldots, n$$

$$\hat{\sigma}_{i,l}^{2} = \frac{1}{S}\sum_{r=1}^{S} (x_{i,r}^{l} - \overline{X}_{i}^{l})^{2} \quad i = 1, \ldots, n$$

$$\hat{\sigma}_{ij,l}^{2} = \frac{1}{S}\sum_{r=1}^{S} (x_{i,r}^{l} - \overline{X}_{i}^{l})(x_{j,r}^{l} - \overline{X}_{j}^{l}) \quad i, j = 1, \ldots, n \quad i \neq j. \tag{3.17}$$

Se pueden encontrar más detalles sobre EMNA$_{global}$ en (Larrañaga y col., 2001). Se han propuesto dos variaciones de este algoritmo, una versión adaptativa, conocida como EMNA$_a$ y otra incremental conocida como EMNA$_i$

EMNA$_a$

EMNA$_a$ (Estimation of Multivariate Normal Algorithm – adaptive) es una adaptación del algoritmo EMNA$_{global}$. La principal característica de esta versión es la forma de obtener el primer modelo $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_1, \Sigma_1)$, que 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 $\boldsymbol{x}_{ge}^{l}$ a partir de la densidad normal multivariante. Si la evaluación del individuo es mejor que la del peor individuo $\boldsymbol{x}^{l,S}$ de la población actual, entonces el individuo simulado sustituye al peor en la población. En este caso, también es necesario volver a estimar la función de densidad normal multivariante. La actualización de la función de densidad se realiza mediante las siguientes fórmulas, que se pueden obtener con métodos algebraicos (Larrañaga y col., 2001):

$$\boldsymbol{\mu}_{l+1} = \boldsymbol{\mu}_{l} + \frac{1}{S}\left(\boldsymbol{x}_{ge}^{l} - \boldsymbol{x}^{l,S}\right) \tag{3.18}$$

$$\begin{aligned}
\sigma_{ij,l+1}^{2} = \sigma_{ij,l}^{2} &- \frac{1}{S^2}\left(x_{ge,i}^{l} - x_{i}^{l,S}\right) \cdot \sum_{r=1}^{S}\left(x_{j}^{l,r} - \mu_{j}^{l}\right) \\
&- \frac{1}{S^2}\left(x_{ge,j}^{l} - x_{j}^{l,S}\right) \cdot \sum_{r=1}^{S}\left(x_{i}^{l,r} - \mu_{i}^{l}\right) + \frac{1}{S^2}\left(x_{ge,i}^{l} - x_{i}^{l,S}\right)\left(x_{ge,j}^{l} - x_{j}^{l,S}\right) \\
&- \frac{1}{S}\left(x_{i}^{l,S} - \mu_{i}^{l+1}\right)\left(x_{j}^{l,S} - \mu_{j}^{l+1}\right) + \frac{1}{S}\left(x_{ge,i}^{l} - \mu_{i}^{l+1}\right)\left(x_{ge,j}^{l} - \mu_{j}^{l+1}\right)
\end{aligned} \tag{3.19}$$

donde $\boldsymbol{x}_{ge}^{l}$ representa a los individuos generados en la l-ésima iteración. La figura 3.10 muestra el pseudocódigo del algoritmo EMNA$_a$.

EMNA$_a$

$D_0 \leftarrow$ Generar R individuos al azar (la población inicial)

Seleccionar $S < R$ individuos de D_0 de acuerdo a un método de selección

Obtener la primera función de densidad normal multivariante $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_1, \Sigma_1)$

for $l = 1, 2, \ldots$ hasta alcanzar un criterio de parada

Generar un individuo $\boldsymbol{x}^l_{ge}$ de $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_l, \Sigma_l)$

if ($\boldsymbol{x}^l_{ge}$ es mejor que el peor individuo, $\boldsymbol{x}^{l,N}$) entonces

1.- Añadir $\boldsymbol{x}^l_{ge}$ a la población, eliminando $\boldsymbol{x}^{l,N}$ de ella

2.- Obtener $\mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}_{l+1}, \Sigma_{l+1}\right)$

Figura 3.10: Pseudocódigo del algoritmo EMNA$_a$.

EMNA$_i$

EMNA$_i$ (Estimation of Multivariate Normal Algorithm – incremental) también genera en cada paso un solo individuo $\boldsymbol{x}^l_{ge}$ a partir de la normal univariante. En este caso, ese individuo siempre se añade a la población, pero sin eliminar ningún individuo. La población se va incrementando constantemente, a diferencia del caso anterior en el que el tamaño de la población permanecía constante. Su principal interés es que las reglas para actualizar la función de densidad son más simples que en EMNA$_a$. Podemos encontrar más detalles sobre este algoritmo en (Larrañaga y col., 2001).

EGNA$_{ee}$, EGNA$_{BGe}$, EGNA$_{BIC}$

En esta sección presentamos tres versiones diferentes del algoritmo EGNA (Estimation of Gaussian Networks Algorithm): EGNA$_{ee}$ (Larrañaga y col., 2000; Larrañaga y Lozano, 2001), EGNA$_{BGe}$ y EGNA$_{BIC}$ (Larrañaga y col., 2001). En todos ellos, para representar el modelo, se utiliza una red Gaussiana. La figura 3.11 muestra el pseudocódigo de los algoritmos EGNA. Los pasos básicos que realizan estos algoritmos son los siguientes:

1. Realizar el aprendizaje estructural de la red Gaussiana utilizando alguno de los métodos propuestos: test de exclusión de arcos, búsquedas con métricas Bayesianas o con la de la máxima verosimilitud penalizada.

2. Estimar los parámetros de la red (por máxima verosimilitud)

3. Generación de la red Gaussiana.

EGNA$_{ee}$, EGNA$_{BGe}$, EGNA$_{BIC}$

For $l = 1, 2, \ldots$ hasta alcanzar un criterio de parada

$D_{l-1}^{S} \leftarrow$ Seleccionar S individuos de D_{l-1}

(i) $\hat{E}_l \leftarrow$ Aprendizaje estructural mediante:
test de exclusión de arcos $\rightarrow$ EGNA$_{ee}$
búsquedas con métricas Bayesianas $\rightarrow$ EGNA$_{BGe}$
búsquedas con la de máxima verosimilitud penalizada $\rightarrow$ EGNA$_{BIC}$

(ii) $\hat{\boldsymbol{\theta}}^{l} \leftarrow$ Estimar los parámetros de $\hat{E}_l$

(iii) $M_l \leftarrow (\hat{E}_l, \hat{\boldsymbol{\theta}}^{l})$

(iv) $D_l \leftarrow$ Muestrear R individuos de M_l utilizando la versión continua del algoritmo PLS

Figura 3.11: Pseudocódigo de los algoritmos EGNA$_{ee}$, EGNA$_{BGe}$, y EGNA$_{BIC}$.

4. Simulación de la función de densidad conjunta expresada por la red Gaussiana aprendida anteriormente. Para ello se puede utilizar una adaptación al entorno continuo del algoritmo PLS descrito en la sección 2.4.

La principal diferencia entre las diferentes versiones del algoritmo EGNA está en la manera de inducir la red Gaussiana: en EGNA$_{ee}$ la red Gaussiana se aprende en cada generación mediante test de exclusión de arcos, mientras que los modelos EGNA$_{BGe}$ y EGNA$_{BIC}$ utilizan métodos de búsqueda + score. EGNA$_{BGe}$ utiliza métricas Bayesianas, de manera que obtiene el mismo valor con redes Gaussianas que reflejan las mismas dependencias, y EGNA$_{BIC}$ utiliza el valor de la máxima verosimilitud penalizada. Estos modelos de inducción se han comentado en la sección 2.6.

Podemos encontrar diversas aplicaciones de estos algoritmos en (Cotta y col., 2001), (Bengoetxea y col., 2001b), (Lozano y col., 2001) y (Robles y col., 2001).

El problema principal en estos casos, en los que se tienen en cuenta múltiples dependencias entre variables, es la cantidad de tiempo de ejecución que consumen estas aproximaciones debido principalmente a la complejidad del modelo gráfico probabilístico que necesitan aprender. La implementación paralela de estos algoritmos ha resultado una forma de eficiente de reducir este tiempo (Mendiburu, 2006).

La mejora y extensión de los EDAs, buscando alternativas más eficientes, es un tema en el que se sigue trabajando actualmente (Sagarna y Lozano, 2006; Mendiburu y col., 2007; Santana y col., 2008), así como sobre las aplicaciones de dichos algoritmos (Romero y Larrañaga, 2009; Zhang y col., 2008; Armañanzas y col., 2008). Podemos

encontrar una revisión del trabajo realizado en este campo durante los últimos años en (Santana y col., 2009).

Capítulo 4

Alternativas en el aprendizaje de los EDAs utilizando clasificadores Bayesianos

Como se ha comentado en el capítulo 3, los EDAs utilizan modelos gráficos probabilísticos para aprender el modelo probabilístico que se utilizará en la generación de nuevos individuos. Este aprendizaje se realiza en cada iteración, y de este modelo se genera una nueva población. En los EDAs, el conjunto de individuos seleccionados para aprender el modelo gráfico probabilístico se compone generalmente de los mejores (los que obtienen mejor valor en la función objetivo). Además, en prácticamente todos los EDAs presentados en la literatura, los valores obtenidos en la función objetivo por cada uno de los individuos seleccionados no se tienen en cuenta a la hora de obtener el modelo, de manera que una vez seleccionados los mejores en D_l^S, no se tienen en cuenta las diferencias entre ellos, considerándolos a todos iguales en el proceso de aprendizaje. Sin embargo, el valor obtenido en la función objetivo por cada uno de los individuos seleccionados, también podría ser considerado en el aprendizaje. A continuación presentamos tres maneras posibles de incluir esto:

- Dar un peso diferente a cada individuo dependiendo de su valor en la función de evaluación.

  Lo que se pretende es tener en cuenta el valor obtenido por cada individuo al ser evaluado por la función objetivo en el proceso de aprendizaje. Una forma de tener en cuenta este valor en la construcción del modelo gráfico probabilístico, se consigue dando un peso diferente a cada uno de los individuos seleccionados en función del valor obtenido al ser evaluado. Un ejemplo de esta idea se utiliza en el algoritmo BSC (Bit-Based Simulated Crossover) (Syswerda, 1993) comentado en la sección 3.2.1. Otra manera de considerar las diferencias en la bondad de los individuos en una población consiste en utilizar un método de selección proporcional, como puede ser la selección basada en la distribución de

Boltzman (Mühlenbein y Mahning, 1999).

- Añadir el valor obtenido en la función objetivo como una nueva variable en el modelo.

  Esta segunda opción añade el valor obtenido por los diversos individuos en la función objetivo como una nueva variable. Esta variable se incluye en el modelo gráfico probabilístico junto con las variables $X_1, \ldots, X_n$. Normalmente, en la mayoría de los casos, el valor que se obtiene en la función objetivo es un valor real. El hecho de incluir este valor como otra variable, requiere que los algoritmos de aprendizaje aplicados sean capaces de tratar con una variable continua, mientras que el resto de las variables pueden ser discretas. Cuando éste es el caso, los métodos de aprendizaje que se pueden utilizar en la construcción del modelo gráfico probabilístico son más complejos y requieren un tiempo considerable de ejecución.

- Convertirlo en un problema de clasificación supervisada.

  La idea principal aquí es clasificar a todos los individuos de una población en diversas clases, y utilizar algoritmos para construir clasificadores Bayesianos que se utilizarán para crear a los nuevos individuos. Estos clasificadores se pueden obtener de manera que consideren las características de los individuos que pertenecen a las clases más aptas y que eviten las de las clases que dan peores resultados. Aquí el problema es seleccionar las clases que se utilizarán para crear el clasificador. Ésta es la opción que se ha elegido para implementar en los nuevos algoritmos que planteamos.

En este capítulo se describe el nuevo método que se ha desarrollado para problemas de optimización y al que hemos llamado Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOAs) (Miquélez y col., 2004; Miquélez y col., 2007). En esta propuesta se combina reconocimiento supervisado de patrones con computación evolutiva. Como se ha comentado anteriormente, los EBCOAs están basados en la utilización de clasificadores Bayesianos en computación evolutiva.

4.1 Utilización de clasificadores en optimización

La clasificación supervisada consiste en construir un modelo de clasificación o clasificador, basándose en la información de un conjunto de datos ya clasificados. Formalmente, el problema de la *clasificación supervisada* con n variables predictoras, consiste en la asignación a un vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{R}^n$ de una de las $|C|$ clases de la variable clase C. Cada una de las posibles clases se denota por medio de c y un *clasificador* se define como una función $\gamma : (x_1, \ldots, x_n) \rightarrow \{1, 2, \ldots, |C|\}$. El objetivo de este problema consiste en encontrar el clasificador que asigne a cada individuo la verdadera clase que le corresponde.

La exactitud de un clasificador es la probabilidad de clasificar correctamente una instancia elegida al azar (Kohavi, 1995). En este sentido, es aconsejable inducir el modelo a partir de un conjunto de datos, llamado conjunto de entrenamiento y aplicar dicho modelo, en la fase de clasificación, a otro conjunto de datos al que llamaremos conjunto de prueba. Una forma de medir la precisión del clasificador es a través del *porcentaje de casos bien clasificados*.

Aunque existen diferentes métodos de validación de un clasificador, en esta tesis se ha utilizado el método de validación cruzada ó *k-fold cross-validation* (Stone, 1974). Este método consiste en dividir el conjunto de datos ya clasificados en k-partes de igual tamaño de manera que $k-1$ partes constituyen el conjunto de entrenamiento y la otra parte se utiliza como conjunto de prueba. Con el conjunto de entrenamiento se induce el clasificador y se aplica al conjunto de prueba para calcular el porcentaje de bien clasificados. Este proceso se repite k veces de manera que en cada ocasión se utilice una partición diferente como conjunto de prueba y las otras $k-1$ restantes como conjunto de entrenamiento. La precisión del clasificador será la media de la precisión de cada uno de los k procesos intermedios. La desviación estándar debería ser aproximadamente la misma, independiente del número de subconjuntos (Kohavi, 1995). El método *leave-one-out* (dejar-uno-fuera) en un caso especial del método de validación cruzada en el que la base de datos es particionada k veces, siendo k igual al número de casos originales n. Este tipo de validación se aplica generalmente a bases de datos con relativamente pocos casos.

Los modelos de clasificación supervisada pueden dividirse en dos categorías dependiendo de la técnica de selección de variables que se aplique: enfoque directo o *wrapper* y enfoque indirecto o *filter*. En la aproximación directa o *wrapper* (Kohavi y John, 1997) cada subconjunto de variables es determinado por un algoritmo de búsqueda, el cual está implícito en la construcción del clasificador. El algoritmo de búsqueda es guiado por una función de mérito o porcentaje de casos bien clasificados obtenida en base a la validación cruzada de k particiones del conjunto de entrenamiento.

En la aproximación indirecta o *filter* (Blum y Langley, 1997; Lewis, 1998) la evaluación se lleva a cabo teniendo en cuenta únicamente las características intrínsecas de los datos sobre la base de la relación de cada una de las variables con la clase. Los distintos criterios de selección de variables basados en aproximaciones indirectas se fundamentan en medidas probabilísticas, guiadas por distancias, inspiradas en la Teoría de la Información, etc.

La información mutua o *mutual information* es una de las medidas filter más ampliamente utilizadas. Esta medida indica la cantidad de incertidumbre que el conocimiento de una variable predictora es capaz de reducir con respecto al estado en el que se encuentre una segunda variable.

Una de las primeras propuestas de aplicación de técnicas de clasificación en optimización es el modelo Learnable Execution Model (LEM) (Michalski, 2000). Al contrario de otras técnicas de computación evolutiva como GAs y EDAs, los algoritmos LEM aplican clasificadores para evolucionar hacia una nueva población de soluciones. En

este método, los individuos de una población se dividen en mejores y peores, de manera que se mantienen las características de los buenos, mientras que se evitan las de los malos. Michalski basa LEM en un método de aprendizaje llamado AQ18 (Kaufman y Michalski, 1999). Este método de clasificación supervisada utiliza métodos generales de aprendizaje de inducción de reglas que son configurables para una convergencia más rápida. LEM puede verse como un acercamiento híbrido que aplica modelos de aprendizaje no estadístico con mecanismos evolutivos de cómputo tradicionales (Ventura y col., 2002).

Existen otros métodos estadísticos que combinan modelos estadísticos de clasificación y computación evolutiva. Como ejemplo de esto tenemos los árboles de decisión (Llorà y Goldberg, 2003; Muñoz, 2003).

4.2 Categorización de clasificadores Bayesianos en base a su complejidad

En la literatura podemos encontrar propuestas de diferentes tipos de clasificadores para combinar con técnicas de computación evolutiva. Uno de los mejores ejemplos es el algoritmo LEM (Michalski, 2000) que emplea reglas para la construcción de un clasificador que identifica las principales diferencias entre el grupo de los mejores y los peores individuos de la población. En los EBCOAs se utilizan, con este mismo objetivo, clasificadores Bayesianos basados en redes Bayesianas y redes condicionales Gaussianas.

A continuación se describen algunos clasificadores Bayesianos que se utilizan habitualmente en problemas de clasificación. La característica principal que los distingue es el número de dependencias entre las variables que se consideran en la construcción de la red Bayesiana. La revisión de estos clasificadores se hace desde el más simple al más complejo. También se proporciona una versión continua de modo que puedan ser aplicados a dominios continuos.

En presencia de variables continuas, una alternativa es asumir que las variables continuas siguen una distribución Gaussiana. En este caso, la estructura que utilizamos se conoce como una red condicional Gaussiana (Geiger y Heckerman, 1994; Lauritzen, 1996; Pérez y col., 2006). Este tipo de red puede tratar con variables continuas y discretas, y, por tanto, es una alternativa para trabajar con variables mixtas sin necesidad de discretizar las variables continuas. Una limitación estructural de las redes condicionales Gaussianas es que una variable discreta no puede tener como padre a una variable continua. Los clasificadores utilizados en este caso se limitan a modelos donde todas las variables predictoras son continuas y la única variable discretas es la variable clase, que es el padre de todas las variables predictoras incluidas en el modelo.

En algún caso, estos clasificadores pueden considerarse demasiado simples o poco eficientes desde el punto de vista de la clasificación. Sin embargo, en nuestro caso, estos clasificadores se utilizan para la optimización con EBCOAs. El propósito, por tanto,

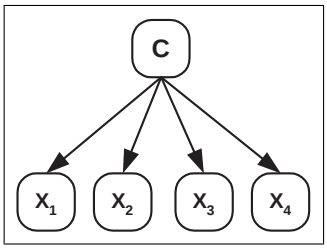

Figura 4.1: Estructura gráfica del modelo naïve Bayes.

es tener un algoritmo de aprendizaje relativamente eficaz que se pueda ejecutar en un tiempo razonable en cada generación.

4.2.1 Naïve Bayes

El paradigma que combina el teorema de Bayes y la hipótesis de independencia condicional dada la clase se conoce como *idiot Bayes* ((Ohmann y col., 1988)), *naïve Bayes* ((Kononenko, 1990)), *simple Bayes* (Gammerman y Thatcher, 1991), o *independent Bayes* (Todd y Stamper, 1994). Aunque tiene una larga tradición en la comunidad de reconocimiento de patrones (*pattern recognition*) (Duda y Hart, 1973), en el campo del aprendizaje automático (*machine learning*), el clasificador naïve Bayes se comenta por primera vez en (Cestnik y col., 1987). Sucesivamente, se ha ido comprobando su potencialidad y robustez en problemas de clasificación supervisada. En ese sentido, naïve Bayes ha demostrado ser eficaz para muchos problemas de clasificación (Domingos y Pazzani, 1997), pudiendo obtener resultados comparables a los obtenidos con otros clasificadores más complejos.

El clasificador naïve Bayes (Minsky, 1961) considera todas las variables $X_1, \ldots, X_n$ condicionalmente independientes dado el valor de la clase C. Por tanto, el modelo gráfico probabilístico corresponde a una estructura fija como la mostrada en la Figura 4.1.

La principal ventaja de la aproximación naïve Bayes consiste en que la estructura siempre es fija. Por tanto, el proceso de aprendizaje del clasificador es muy rápido, ya que la única tarea es estimar los parámetros que más tarde deberá considerar esta red Bayesiana.

Siguiendo el modelo naïve Bayes, al clasificar un ejemplo $\boldsymbol{x}$, se le asignará la clase c con una determinada probabilidad. Esta probabilidad se calcula aplicando:

$$p(c \mid \boldsymbol{x}) \propto p(c, \boldsymbol{x}) = p(c) \prod_{i=1}^{n} p(x_i|c). \tag{4.1}$$

La estimación de la probabilidad de la clase, $p(c)$, así como de las probabilidades condicionales, $p(x_i|c)$, se realiza a partir de los individuos seleccionados en cada generación.

En el dominio continuo, es habitual asumir que la función de densidad conjunta sigue una distribución normal n-dimensional, y debido a la consideración de independencia entre las variables, dada la variable clase C, ésta se factoriza como un producto de densidades condicionales normales y univariantes. Por tanto, al clasificar a un nuevo individuo mediante el clasificador naïve Bayes tenemos que:

$$p(C = c|X_1 = x_1, \ldots X_n = x_n) \propto$$
$$p(c) \cdot f(x_1|c) \cdot f(x_2|c) \cdot \ldots \cdot f(x_n|c)$$

donde

$$f(x_i|c) = \frac{1}{\sqrt{2\pi}\sigma_{ic}} e^{-\frac{1}{2}(\frac{x_{ic}-\mu_{ic}}{\sigma_{ic}})^2}$$

para todo $i = 1, \ldots, n$ y $c = 1, \ldots, |C|$ donde μ_{ic} y σ_{ic} representan la media y la desviación estándar de $X_i|C = c$ respectivamente.

Para aplicar un clasificador naïve Bayes en los EBCOAs, la estimación de la probabilidad a priori de la clase, $p(c)$, así como de los parámetros μ_{ic} y σ_{ic} de las funciones de densidad condicionales, $f(x_i|c)$, se realiza en base a los individuos seleccionados en cada generación.

Nótese que este tipo de clasificador Bayesiano es el más indicado para problemas de optimización en los cuales las variables que forman los individuos son realmente independientes dada la clase.

4.2.2 Selective naïve Bayes

La principal diferencia entre el modelo *selective naïve Bayes* (Kohavi y John, 1997; Langley y Sage, 1994) y *naïve Bayes* es que en el *selective naïve Bayes* no todas las variables tienen que estar presentes en el modelo final. Tal y como se comenta en la sección 4.3.4, en determinados problemas de clasificación podemos encontrar lo que llamamos variables irrelevantes, es decir, variables cuyas distribuciones de probabilidad condicionadas a cualquier valor de la variable clase permanecen iguales, o variables redundantes, aquellas cuya correlación con otras variables predictoras es muy alta. Para este tipo de problemas, la condición impuesta por el modelo *naïve Bayes* de tener que considerar todas las variables, puede resultar muy estricta. El comportamiento de naïve Bayes no es muy bueno cuando hay variables redundantes (Liu y Motoda, 1998; Inza y col., 2000). Teniendo en cuenta todo esto, el propósito del clasificador *selective naïve Bayes* es mejorar el rendimiento del clasificador naïve Bayes eliminando las variables irrelevantes y redundantes, considerando solo aquellas que proporcionan mayor información. La Figura 4.2 muestra una posible estructura de red Bayesiana para un problema de cuatro variables, donde una de ellas, X_3, no aparece en la estructura final.

El algoritmo comienza inicializando el conjunto de variables como conjunto vacío y evalúa la exactitud del clasificador resultante para su posterior comparación. Para la validación del clasificador se utiliza validación cruzada (*k-fold cross-validation*) con

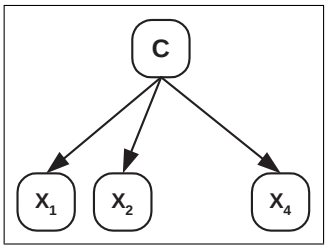

Figura 4.2: Ejemplo de estructura gráfica del modelo *selective naïve Bayes* para un problema de cuatro variables.

$k = 10$. En cada iteración, el método considera añadir cada variable no utilizada al conjunto y evalúa la exactitud del clasificador resultante. Se añade aquella variable que obtiene un clasificador con mejor porcentaje de bien clasificados. El algoritmo termina cuando no existe ninguna variable que al añadirla mejore la exactitud del clasificador.

Siguiendo el modelo selective naïve Bayes y usando la estructura mostrada en la figura 4.2, a un individuo $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ se le puede asignar la clase

$$c^* = \arg \max_c p(c)p(x_1|c)p(x_2|c)p(x_4|c) \tag{4.2}$$

4.2.3 Semi–naïve Bayes

El clasificador *semi–naïve Bayes* es un clasificador Bayesiano que pretende mejorar al clasificador naïve Bayes evitando las estrictas restricciones de este modelo y permitiendo agrupar algunas de las variables, teniendo en cuenta dependencias entre conjuntos de variables. Las variables relacionadas se representan como un solo nodo en la red Bayesiana o en caso de valores continuos, en la red condicional Gaussiana.

En 1991 Kononenko (Kononenko, 1991) propone un método para la identificación de atributos que son condicionalmente independientes basado en una prueba estadística que determina la probabilidad de que dos atributos no sean independientes. El algoritmo une dos atributos si hay una probabilidad mayor que 0,5 de que los atributos no sean independientes. Posteriormente, Pazzani (Pazzani, 1997) presenta dos métodos alternativos para detectar dependencias entre variables: *FSSJ (Forward Sequential Selection and Joining.- Selección secuencial hacia delante y unión)* y *BSEJ (Backward Sequential Elimination and Joining.- Eliminación secuencial hacia atrás y unión)*. Las variables dependientes se agrupan en una nueva variable multidimensional que se considera como un solo nodo de la red. Las figuras 4.3 y 4.4 muestran el pseudocódigo del algoritmo *FSSJ* y *BSEJ* respectivamente. El algoritmo se guía por una métrica wrapper 10-*fold cross–validation* o bien *leave one out* dependiendo del tamaño de la base de datos.

Tal y como puede verse en la Figura 4.3 el algoritmo *FSSJ* efectúa una modelización voraz hacia adelante guiado por la estimación del porcentaje de casos bien clasificados.

Inicializar el conjunto de variables a utilizar como vacío
Clasificar todos los ejemplos como pertenecientes a la clase de mayor $p(c)$
Repetir en cada iteración:
  Elegir la mejor opción entre
  (a) Considerar cada variable no utilizada por el clasificador
    como una nueva variable a incluir en el modelo,
    como condicionalmente independiente, dada la clase,
    de las otras variables usadas por el clasificador
  (b) Considerar la unión de cada variable no usada en el modelo actual
    con cada variable usada actualmente
  Evaluar cada opción utilizando la estimación del porcentaje de
    casos bien clasificados
Hasta que no se obtenga ninguna mejora

Figura 4.3: Pseudocódigo del algoritmo *FSSJ* utilizado para construir el modelo *semi–naïve Bayes*.

Comenzar considerando todas las variables como condicionalmente independientes dada la variable clase
Repetir en cada iteración:
  Elegir la mejor opción entre
  (a) Considerar reemplazar cada par de variables utilizadas por el clasificador como una nueva variable que sea el producto cartesiano de ambas
  (b) Considerar eliminar cada variable utilizada por el clasificador
  Evaluar cada opción utilizando la estimación del porcentaje de casos bien clasificados
Hasta que no se obtenga ninguna mejora

Figura 4.4: Pseudocódigo del algoritmo *BSEJ* utilizado para construir el modelo *semi–naïve Bayes*.

Comienza considerando como modelo inicial la regla simple que consiste en clasificar todos los ejemplos, independientemente de sus características, como pertenecientes a la clase más numerosa. A continuación, mientras se vaya mejorando la estimación del porcentaje de bien clasificados, se va efectuando en cada paso la mejor opción entre incluir en el modelo una variable de las que todavía no formaban parte del mismo, u obtener una nueva variable como producto cartesiano entre alguna de las variables (o supervariables[1]) ya incluidas en el modelo y la que acaba de incluirse.

[1]La denominación supervariable hace alusión a la variable resultante del producto cartesiano entre

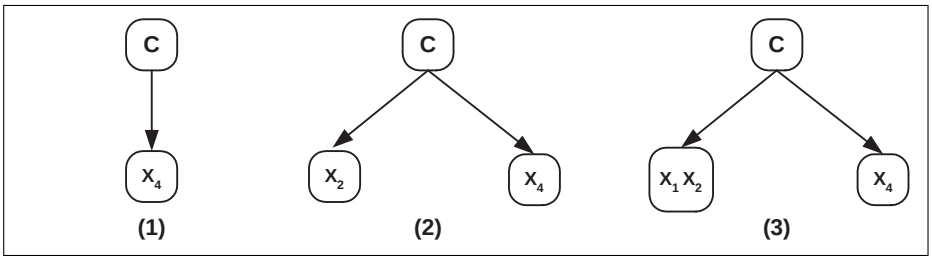

Figura 4.5: Pasos para la construcción de un clasificador Bayesiano *semi–naïve Bayes* siguiendo el algoritmo *FSSJ*, para un problema de cuatro variables. X_1, X_2, X_3, X_4 son las variables predictoras y C la variable a clasificar.

La figura 4.5 muestra un ejemplo de aplicación del algoritmo *FSSJ*. Es importante mencionar que el clasificador *semi–naïve Bayes* también contempla la posibilidad de no incluir algunas variables en el modelo gráfico final, indicando que estas variables no son relevantes para la asignación a una u otra clase. Esto es lo que ocurre en este ejemplo con la variable X_3. Como resultado, siguiendo el modelo semi–naïve Bayes y usando el clasificador obtenido en esta figura, a un individuo $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ le podemos asignar la siguiente clase:

$$c^* = \arg\max_c p(c)p(x_1, x_2|c)p(x_4|c). \tag{4.3}$$

En el caso de variables continuas, cada una de las variables agrupadas se trata como una sola supervariable en cuanto a la factorización de la distribución de probabilidad. Para el caso del ejemplo anterior de la Figura 4.5(3) con variables continuas, un individuo $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ será asignado a la siguiente clase:

$$c^* = \arg\max_c p(c)f(x_1, x_2|c)f(x_4|c). \tag{4.4}$$

En los casos en los que una variable X_i se estima como independiente del resto (como la variable X_4 en el último ejemplo), $f(x_i|c)$ puede calcularse como:

$$f(x_i|c) = \frac{1}{\sqrt{2\pi}\sigma_{ic}}\ e^{-\frac{1}{2}(\frac{x_{ic}-\mu_{ic}}{\sigma_{ic}})^2}$$

y análogamente, para el caso de dependencias con un número p de variables, como las variables X_1 y X_2 en el ejemplo, se supondrá que el factor correspondiente sigue una distribución normal p-dimensional para cada valor de la variable C. Considerando que $\mathbf{z}_j$ representa un valor del j-ésimo grupo de p variables, tendríamos que

$$f(\mathbf{z}_j|c) = \frac{1}{\sqrt{(2\pi)^p|\Sigma_{jc}|}}\ e^{-\frac{1}{2}(\mathbf{z}_j-\mu_{jc})^T\Sigma_{jc}^{-1}(\mathbf{z}_j-\mu_{jc})}$$

dos o más variables originales.

Calcular $I(X_i, X_j \mid C)$ para cada par de variables predictoras, con $i \neq j$
Construir un grafo no dirigido, donde los nodos corresponden a las variables predictoras $X_1, \ldots, X_n$, asignando el peso $I(X_i, X_j \mid C)$ a la arista que conecta X_i y X_j
Asignar la arista con mayor peso al árbol que estamos construyendo

Repetir en cada iteración:
    Examinar la arista de mayor peso y añadirla al árbol, a no ser que esto forme un ciclo
    En ese caso, desecharla y examinar la siguiente arista de mayor peso
Hasta añadir $n - 1$ aristas a la estructura

Transformar el grafo no dirigido en uno dirigido, escogiendo una variable raíz al azar
Construir la estructura *tree augmented naïve Bayes* añadiendo un nodo etiquetado como C, y un arco desde C a cada una de las variables predictoras X_i $(i = 1, \ldots, n)$

Figura 4.6: Pseudocódigo del algoritmo *tree augmented naïve Bayes*.

donde Σ_{jc} es una matriz $p \times p$ que representa la matriz de varianzas-covarianzas del j-ésimo grupo de p variables cuando $C = c$ y μ_{jc} representa su correspondiente esperanza matemática.

4.2.4 Tree augmented naïve Bayes

Otro clasificador Bayesiano que también tiene en cuenta dependencias entre variables, aunque de manera diferente al clasificador anterior, es el denominado *tree augmented naïve Bayes* (*TAN*) (Friedman y col., 1997). Este método construye una estructura en forma de árbol, según se ilustra en la Figura 4.6.

El proceso para crear esta estructura se compone de dos fases: En primer lugar se buscan las dependencias entre las diferentes variables $X_1, \ldots, X_n$. Para ello, el algoritmo utiliza un método basado en la teoría de la información, donde el peso que se asigna a una rama (X_i, X_j) del árbol, viene definido por el valor de la información mutua condicionada a la variable clase

$$\begin{aligned} I(X_i, X_j|C) &= \sum_c p(c) I(X_i, X_j|C = c) \\ &= \sum_c \sum_{x_i} \sum_{x_j} p(x_i, x_j, c) \log \frac{p(x_i, x_j|c)}{p(x_i|c)p(x_j|c)} \end{aligned} \tag{4.5}$$

con $i < j, j = 2, \ldots, n$.

Cuando las variables predictoras, $X_1, \ldots, X_n$, toman valores reales, el procedimiento es una adaptación del algoritmo de Chow-Liu (Chow y Liu, 1968) para variables predictoras continuas, que calcula la información mutua entre dos distribuciones normales univariantes (Cover y Thomas, 1991) a partir de la expresión siguiente:

$$I(X_i, X_j \mid C) = -\frac{1}{2} \sum_{c=1}^{|C|} p(c) \log \left(1 - \rho_c^2(X_i, X_j)\right) \tag{4.6}$$

con $i < j, j = 2, \ldots, n$ donde $\rho_c^2(X_i, X_j) = \frac{Cov_c(X_i, X_j)}{\sqrt{Var_c X_i Var_c X_j}}$ es el coeficiente de correlación entre X_i y X_j.

Con estos valores de la información mutua, el algoritmo construye una estructura en forma de árbol. En una segunda fase la estructura se aumenta a una red Bayesiana añadiendo un arco desde la variable C al resto de variables predictoras $X_1, \ldots, X_n$.

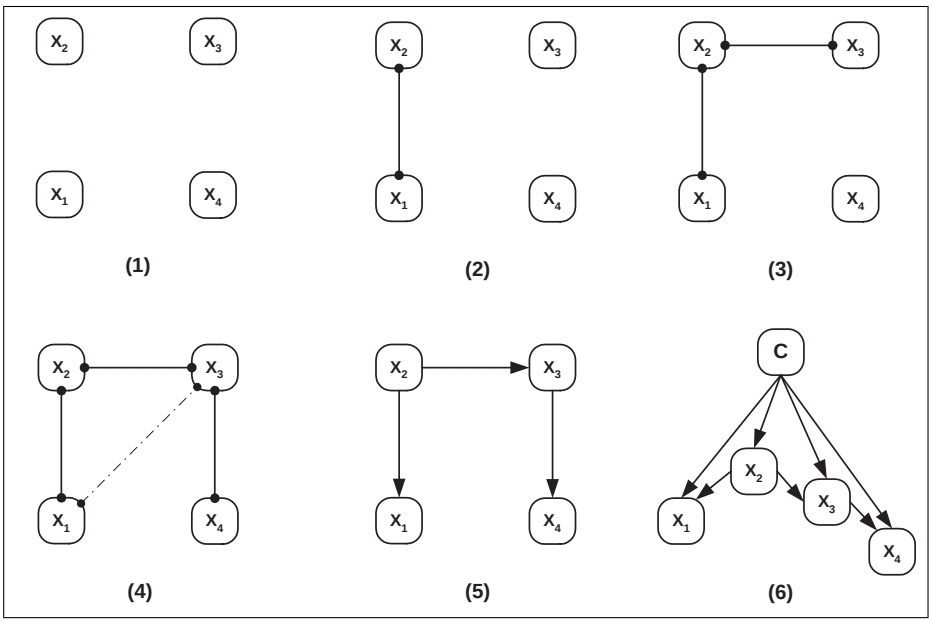

Figura 4.7: Representación de los pasos para la construcción de un clasificador *tree augmented naïve Bayes* para un problema de cuatro variables. X_1, X_2, X_3, X_4 son las variables predictoras y C es la variable a clasificar.

La figura 4.7 muestra un ejemplo de aplicación del algoritmo *tree augmented naïve Bayes*. En este ejemplo, se asume que $I(X_1, X_2|C) > I(X_2, X_3|C) > I(X_1, X_3|C) > I(X_3, X_4|C) > I(X_2, X_4|C)$, $I(X_1, X_4|C)$. En (4) se ha rechazado la arista (X_1, X_3) por formar un ciclo. Finalmente, (6) representa la estructura de árbol aumentado al aplicar la segunda fase del método. Siguiendo el método *tree augmented naïve Bayes*, y

utilizando la estructura representada en esta figura, a un individuo $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ podemos asignarle la clase:

$$c^* = \arg\max_c p(c)p(x_1|c, x_2)p(x_2|c)p(x_3|c, x_2)p(x_4|c, x_3). \tag{4.7}$$

A diferencia de la aproximación *wrapper* utilizada para medir la bondad de la estructura obtenida en el método *semi–naïve Bayes* comentado en el apartado anterior, el algoritmo *tree augmented naïve Bayes* sigue un método *filter* basado en la cantidad de información mutua, donde sólo se consideran dependencias entre pares de variables.

El caso anterior se ha considerado suponiendo que las variables predictoras toman valores discretos. Si consideramos el mismo ejemplo de la figura 4.7 pero para variables continuas, a un individuo $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ se le asignará la clase:

$$c^* = \arg\max_c \quad p(c) \cdot f(x_1|c, x_2) \cdot f(x_2|c) \cdot f(x_3|c, x_2) \cdot f(x_4|c, x_3). \tag{4.8}$$

En este caso, el cálculo de $f(x_i|c, x_{k(i)})$, donde $X_{k(i)}$ representa la variable padre de la predictora X_i en caso de que exista, puede hacerse como

$$\begin{aligned} f(x_i|c, x_{k(i)}) &= \frac{p(c) \cdot f(x_i, x_{k(i)}|c)}{p(c) \cdot f(x_{k(i)}|c)} \\ &= \frac{f(x_i, x_{k(i)}|c)}{f(x_{k(i)}|c)}. \end{aligned} \tag{4.9}$$

4.2.5 Otros métodos

Además de los clasificadores presentados anteriormente, existen otros métodos de construcción de clasificadores Bayesianos teniendo en cuenta más o menos dependencias entre variables. En los últimos años se han propuesto diferentes métodos, ya que constituyen un tema de investigación importante. Como ejemplos de estos clasificadores Bayesianos podemos mencionar el clasificador Bayesiano K-dependencias (Sahami, 1996), redes Bayesianas generales (Neapolitan, 2003), y multiredes Bayesianas (Kontkanen y col., 2000). Sin embargo, estos clasificadores, debido a su alto coste computacional, no se han utilizado en los algoritmos presentados en este capítulo, por lo que no entramos en su descripción.

4.3 Descripción del nuevo método

En esta sección se describe el nuevo método propuesto al que, como ya se ha comentado, hemos denominado *Evolutionary Bayesian Classifier-based Optimization Algorithms* (*EBCOAs*). En este método se sigue un proceso de cálculo evolutivo similar a los EDAs, aunque se diferencian principalmente en el método que se utiliza para

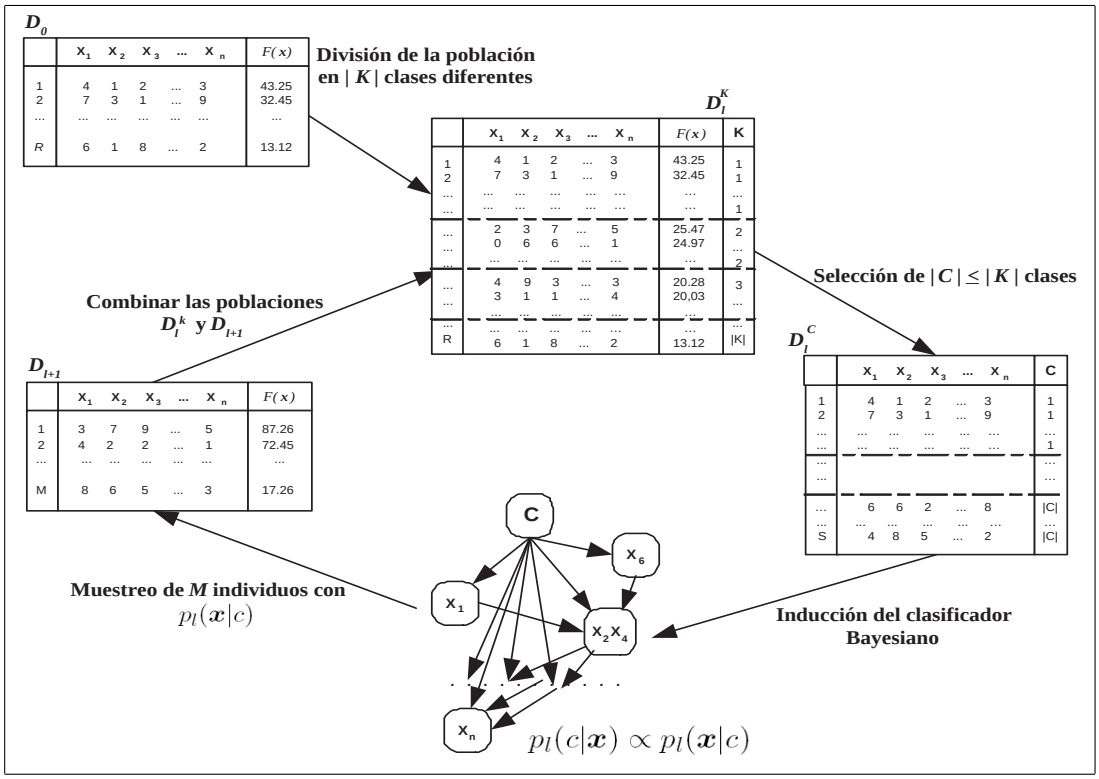

Figura 4.8: Ilustración del método EBCOA en el proceso de la optimización.

construir la red Bayesiana o Gaussiana. Mientras que en los EDAs se utilizan algoritmos de inducción de redes Bayesianas para el aprendizaje, los EBCOAs aprenden el modelo mediante clasificadores Bayesianos que utilizan la información proporcionada por el valor obtenido en cada caso en la función objetivo. Además, a la hora de construir este clasificador, los EBCOAs tienen en cuenta las características, no solo de los mejores individuos de la población actual, como se hace en los EDAs, sino también las de los peores. De esta manera, el modelo gráfico probabilístico representa las características de los mejores y de los peores individuos. Esta idea tiene como objetivo evitar una convergencia demasiado rápida que en algunos problemas de optimización lleva a los EDAs a caer en óptimos locales y no alcanzar el resultado deseado.

En la figura 4.8 se representa el método seguido por los EBCOAs. Si comparamos esta figura con la figura 3.2, podemos observar que las diferencias entre EBCOAs y EDAs están precisamente en el paso de aprendizaje del modelo.

4.3.1 Notación

Sea D_l la l-ésima población de R individuos que tiene que evolucionar hacia la población $(l+1)$-ésima. En los EBCOAs, antes de realizar el aprendizaje, la población D_l se divide en $|K|$ clases diferentes, tratando de plantear el problema desde

una perspectiva de clasificación supervisada. La variable K puede tomar los valores $\{1, 2, \ldots, |K|\}$. Representamos por D_l^K a la población D_l después de dividirla en $|K|$ clases, de manera que a cada individuo de la población le hemos asignado un valor k de la variable K con $1 \leq k \leq |K|$. Dicho valor k representa la clase a la que se ha asignado cada individuo. Ya que no todas las clases se utilizan en el aprendizaje, antes de obtener el clasificador Bayesiano, elegimos los individuos de $|C| \leq |K|$ clases y el resto sencillamente los ignoramos, para el proceso de aprendizaje. Llamaremos D_l^C al subconjunto de D_l^K que utilizaremos para el aprendizaje. Utilizaremos C para referirnos a la variable clase, con valores denotados por c con $1 \leq c \leq |C|$.

El resultado del paso de aprendizaje en EDAs es construir un modelo gráfico probabilístico, es decir, una red Bayesiana si trabajamos con valores discretos o una red Gaussiana cuando se refiere a valores continuos. En los EBCOAs, esta red es un clasificador Bayesiano construido a partir de las variables $X_1, \ldots, X_n$ y de la variable C.

La tarea principal en los EBCOAs consiste en estimar la $p_l(\boldsymbol{x} \mid c)$, es decir, la probabilidad de que un individuo $\boldsymbol{x}$ pertenezca a cada una de las clases $1, 2, \ldots, |C|$ en D_l^C. Esta probabilidad se debe estimar cada generación, puesto que la población, y por lo tanto la naturaleza de las clases, es diferente en cada una de ellas. En los EBCOAs, la estructura de la red Bayesiana que se obtiene como resultado del paso de aprendizaje, contendrá, igual que en el caso de los EDAs, las variables $X_1, \ldots, X_n$, pero además, la nueva variable C definida anteriormente. Esta variable C estará presente en todas las estructuras obtenidas utilizando algoritmos de construcción de clasificadores Bayesianos por los EBCOAs, y C será siempre la variable padre del resto de variables.

4.3.2 Descripción de los principales pasos de los EBCOAs

En esta sección se describen los principales pasos del método EBCOA, así como las implicaciones de las diversas opciones que se harán en cada uno de ellos.

1. En primer lugar, se genera la población inicial D_0 formada por R individuos. Estos R individuos se generan normalmente siguiendo una distribución uniforme en cada variable. A continuación se evalúa cada individuo.

2. La población D_l se subdivide en un número $|K|$ de clases en función del valor obtenido por los individuos al ser evaluados. Como resultado de este paso se obtiene D_l^K.

3. Ya que las diferencias entre los individuos de las K clases, normalmente no son lo suficientemente grandes como para permitir la construcción de un clasificador, se selecciona un subconjunto $C \subseteq K$ de clases, y el resto son simplemente ignoradas. La población resultante se denota por D_l^C.

4. Con los individuos seleccionados en el paso anterior, se construye el clasificador Bayesiano n–dimensional que mejor refleje las diferencias entre las clases

EBCOAs

$D_0 \leftarrow$ Generar R individuos (población inicial) al azar
Repetir Para cada $l = 1, 2, \ldots$ hasta satisfacer un criterio de parada
$D_l^K \leftarrow$ Dividir los R individuos de D_l en $|K| < R$ clases diferentes según un criterio de clasificación
$D_l^C \leftarrow$ Seleccionar los individuos de $|C| \leq |K|$ clases de D_l^K, que se utilizarán para construir el clasificador Bayesiano. Los individuos de las clases no seleccionadas se ignoran
$p_l(c|\boldsymbol{x}) \propto p_l(\boldsymbol{x}|c) \leftarrow$ Estimar la distribución de probabilidad de cada individuo en D_l^C de pertenecer a cualquiera de las $|C|$ posibles clases
$D_l \leftarrow$ Crear R nuevos individuos (la nueva población) a partir de $p_l(\boldsymbol{x}|c)$

Figura 4.9: Pseudocódigo del método EBCOA.

seleccionadas (por lo general, entre los mejores y peores). La estructura de este clasificador Bayesiano contiene a la variable C como padre de todas las demás, además de todas, o algunas, de las variables predictoras $X_1, X_2, \ldots, X_n$.

5. Por último, se construye la nueva población D_{l+1}. Esta nueva población D_{l+1} se obtiene combinando los M nuevos individuos obtenidos mediante la simulación de la distribución de probabilidad aprendida en el paso anterior y algunos individuos representativos de las diferentes clases de D_l^K, con el fin de obtener un total de R individuos para la próxima generación. Este proceso lo explicamos en la sección 4.3.3.

Los pasos 2, 3, 4 y 5 se repiten hasta alcanzar un criterio de parada. La figura 4.9 muestra el pseudocódigo de este proceso.

Como se puede comprobar, el proceso es similar al descrito en la sección 3.1 para los EDAs. Se diferencia en la forma en que se realiza la selección de los individuos. Mientras en los EDAs se selecciona a los mejores individuos de cada población, los EBCOAs clasifican a estos en diferentes clases para luego seleccionar a los individuos de determinadas clases. Además, a la hora de estimar la distribución de probabilidad, los EBCOAs incluyen la variable clase dentro del modelo.

Clasificación supervisada: etiquetado de los individuos y selección de las clases

En primer lugar se clasifica toda la población en un número fijo $|K|$ de clases diferentes. Estas clases se forman dividiendo a la población entera en grupos de individuos desde los mejores, teniendo en cuenta el criterio de evaluación, hasta los peores. El resultado de este proceso es asignar a cada uno de los R individuos en D_l una etiqueta

k (con $k \in \{1, 2, \ldots, K\}$), y de esta forma se crea la variable clase K en la base de datos D_l^K.

Como en los EBCOAs el objetivo es también considerar las características principales que distinguen a los mejores individuos de los peores, podríamos desechar algunas de las clases D_l^K para facilitar el aprendizaje. Un ejemplo de esta idea es no tener en cuenta las clases intermedias en D_l^K a la hora de aprender el clasificador Bayesiano, para realzar las diferencias entre las clases más distantes, aunque también existen otras opciones tal y como se comenta en la sección 4.3.3. D_l^C es el resultado de eliminar de D_l^K las clases que no se utilizan en el aprendizaje, y C es la variable clase que se utiliza como nodo raíz en el aprendizaje del clasificador Bayesiano, con $|C| \leq |K|$.

El aprendizaje: construcción del clasificador Bayesiano

Mientras que en los EDAs el aprendizaje se realiza mediante la inducción del modelo probabilístico a partir de los mejores individuos de cada población, los EBCOAs utilizan clasificadores Bayesianos para crear el modelo probabilístico, utilizando para ello a los individuos de las diferentes clases que forman la variable C, de manera que dicha variable se incluye como raíz en el modelo. Por lo tanto, el modelo gráfico probabilístico obtenido contendrá un máximo de $n + 1$ nodos (las variables X_1 a X_n y C), siendo siempre la variable C la raíz y el padre de todas las demás. Como resultado de este procedimiento se aprende la distribución de probabilidad, que se puede representar por $p_l(c|\boldsymbol{x}) \propto p_l(c) \cdot p_l(\boldsymbol{x}|c)$ en el caso de variables discretas y por una factorización en la forma $p_l(c|\boldsymbol{x}) \propto p_l(c) \cdot f_l(\boldsymbol{x}|c)$ para variables continuas. Los algoritmos de construcción de clasificadores que se han utilizado para esta fase se han explicado de manera detallada en la sección 4.2.

La complejidad del clasificador Bayesiano utilizado en el proceso de aprendizaje de los EBCOAs, es decir, el grado de dependencias entre las variables $X_1, X_2, \ldots, X_n$ y C que es capaz de tener en cuenta, determina la capacidad del clasificador para reflejar mejor las diferencias entre los mejores y peores individuos de la población.

El paso más costoso para los EBCOAs consiste en estimar satisfactoriamente la distribución de probabilidad $p_l(c|\boldsymbol{x})$. Es importante destacar que en nuestro caso no estamos interesados en la obtención del mejor clasificador Bayesiano posible, sino en obtener un clasificador estrictamente correcto. Estos algoritmos para obtener clasificadores óptimos en forma de red Bayesiana (o red condicional Gaussiana en el caso de variables continuas) consumen mucho tiempo, y el requisito del tiempo de ejecución es crucial en los EBCOAs. Hay que tener en cuenta que este paso de aprendizaje (es decir el paso de construcción del clasificador) se va a aplicar en cada generación. Es más importante utilizar un constructor del clasificador Bayesiano que devuelva un clasificador satisfactorio en un tiempo razonable, que utilizar demasiado tiempo en obtener un clasificador perfecto que ya no tendrá ningún sentido en la siguiente generación.

La simulación: generando la nueva población

El paso de muestreo del modelo gráfico probabilístico para obtener la nueva población D_{l+1} se realiza de una manera similar a como se hace en los EDAs (ver sección 3.1), aunque hay una diferencia importante debido a la existencia de la variable C en el modelo. Cada individuo se genera usando un criterio específico, como por ejemplo la distribución de probabilidad $p_l(\boldsymbol{x}|c)$. Por lo tanto, la simulación del individuo se realiza a partir de la distribución de probabilidad aprendida en el paso anterior.

La diferencia principal viene de la necesidad de reflejar las características que diferencian a los individuos de las mejores y peores clases. En ese sentido, al realizar la simulación para crear la nueva generación de individuos que formarán la siguiente población D_{l+1}, los individuos se instancian utilizando todas las clases en C. Se generan M nuevos individuos, donde $M \leq R$, siendo R el tamaño de la población, ya que en algunos casos es interesante mantener individuos de la generación anterior. Los nuevos individuos se generan utilizando un número diferente de representantes de cada clase $c \in C$ de D_l^C mediante la instanciación proporcional a la distribución de probabilidad de todas las clases $p(c)$, esto es

$$p(c) \propto \sum_{\boldsymbol{x}|\ C(\boldsymbol{x})=c} F(\boldsymbol{x}) \tag{4.10}$$

donde $F(\boldsymbol{x})$ es el valor obtenido al evaluar al individuo $\boldsymbol{x}$, y $C(\boldsymbol{x})$ es la clase asignada al individuo $\boldsymbol{x}$ en D_l^C.

La razón de hacer la simulación de esta manera es asegurarse de que en la siguiente generación estarán presentes las características de todos los individuos de las clases seleccionadas, aunque en mayor proporción, aquellos individuos que obtienen una mejor clasificación en la función objetivo. Después de este proceso, en las nuevas generaciones, se incluirán también individuos de las clases con peor valor en la función objetivo y este hecho asegura que se mantengan presentes las diferencias entre los mejores y peores individuos de las generaciones anteriores del proceso de búsqueda para que los algoritmos puedan converger a la solución óptima. Es importante guardar estas diferencias puesto que la convergencia del conjunto al óptimo, se basa en la capacidad del clasificador Bayesiano de modelar las características principales que hacen que un individuo esté en una u otra clase.

4.3.3 Evolución de la población

Una vez generados los M nuevos individuos que formarán la nueva población D_{l+1} mediante la simulación del modelo creado, existen diferentes opciones para formar la población de nueva generación D_{l+1}^K.

En los EDAs, esta evolución suele realizarse utilizando lo que normalmente se conoce como *elistismo* y que consiste en generar M nuevos individuos, donde $M = R - 1$, de manera que la nueva población estará formada por el mejor de la población

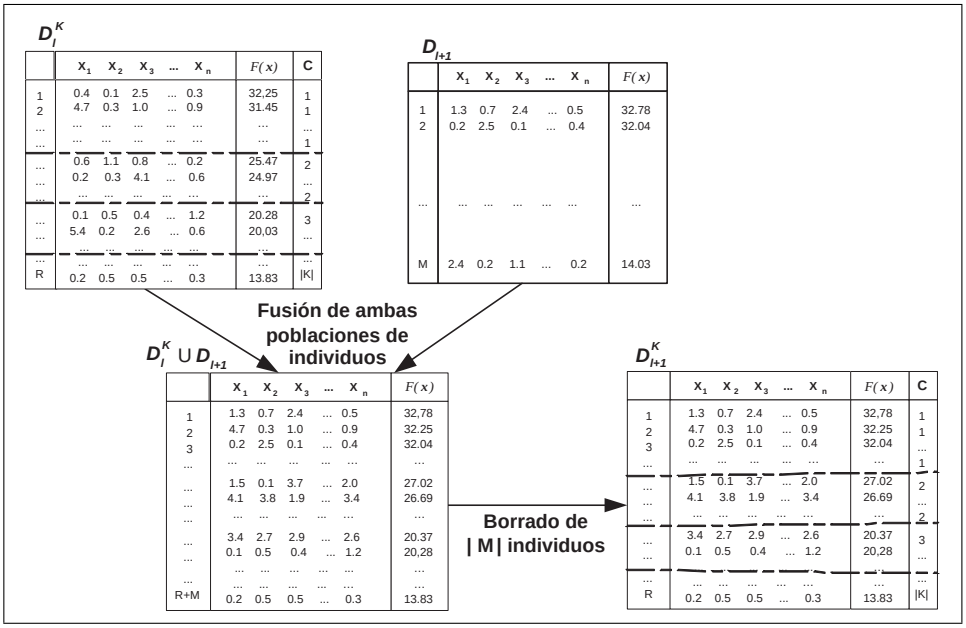

Figura 4.10: Ilustración del proceso de evolución de la población de una generación a la siguiente en el método EBCOA. Transición entre D_l^K y D_{l+1} a D_{l+1}^K.

anterior y los M nuevos individuos generados. Sin embargo, esta técnica tiende a generar poblaciones muy homogéneas.

En los EBCOAs intentamos mantener información de las poblaciones anteriores y crear una población heterogénea. El paso inicial para la creación de D_0^K a partir de la población inicial D_0 se realiza directamente asignando la clase c a cada individuo en función de su correspondiente valor individual en la población, tal y como se describe en la sección anterior. Esta es la alternativa más lógica para construir las diferentes clases.

Sin embargo, en las generaciones sucesivas, creamos M nuevos individuos. En esta etapa, intentamos mantener también información de algunos de los individuos de la generación anterior D_l^K. Hay que tener en cuenta que D_l^C simplemente sirve como un paso intermedio para la fase de aprendizaje de los EBCOAs. La inclusión de un mínimo número de individuos de la población anterior D_l^K tiene como objetivo mantener algunas de las características de los individuos peor valorados dentro del espacio de búsqueda visitado en las generaciones anteriores. Este hecho nos aconseja combinar individuos de D_l^K (donde $|D_l^K| = R$) y los nuevos individuos recientemente generados en la población D_{l+1} (donde $|D_{l+1}| = M$), con el fin de formar a la población D_{l+1}^K, que constituyen la población de la $(l+1)$-ésima generación (también con $|D_{l+1}^K| = R$). La figura 4.10 muestra un ejemplo de este proceso.

Por otra parte, la combinación de las poblaciones debe hacerse de manera que también se mantenga al menos un número mínimo de los individuos mejor valorados

tanto de D_l^K como de D_{l+1}. El método EBCOA obtiene mejores resultados cuando en D_{l+1}^K se mantienen los individuos que mejor se adaptan a las características de cada una de las $|K|$ clases. Hay que tener en cuenta que en cada generación, la población D_l^K mantiene el mismo tamaño de R individuos, y por lo tanto, cada una de las $|K|$ clases en la que se divide esta población contiene el mismo número de individuos en cada generación.

La manera de combinar los individuos de D_l^K y D_{l+1} con el fin de formar la nueva población D_{l+1}^K influye directamente en el rendimiento de los EBCOAs, ya que se eliminará un total de M individuos de estas dos poblaciones. Puesto que la convergencia de los EBCOAs hacia una mejor población depende totalmente de la capacidad del proceso de aprendizaje de proporcionar una estructura que representa las diferencias entre las mejores y peores clases, esta convergencia dependerá de la complejidad del problema de optimización a resolver, del número de clases ($|K|$ y $|C|$) consideradas, y del tamaño de la población (R). Desde nuestra experiencia hemos llegado a la conclusión de que lo más aconsejable es garantizar que por lo menos los mejores individuos de D_l^K y D_{l+1} se conserven en D_{l+1}^K, a fin de que la clase de los mejores de la próxima generación esté compuesta como mínimo, de los mejores individuos encontrados durante todo el proceso de búsqueda hasta el momento. Por lo tanto, es importante definir los valores que deben tomar $|K|$ y $|C|$ así como el paso de selección de las clases para la transición de D_l^K a D_l^C. Nosotros proponemos establecer el número inicial de clases $|K| = 3$. La razón para establecer el número inicial de clases $|K| = 3$ es aumentar las diferencias entre los individuos de las clases seleccionadas. De esta manera tenemos que $|C| = 2$, y, por lo tanto, debemos asegurarnos de que los individuos de las dos clases sean lo suficientemente diferentes como para que el clasificador Bayesiano pueda identificar las características que los diferencian.

Es importante garantizar que los individuos de cada una de las C clases de D_l^C se incluyan en D_{l+1}, siguiendo el principio de proporcionalidad descrito en la ecuación 4.10. Dado que en nuestro caso tenemos dos clases en D_l^C, el número de individuos que se instancia para cada una de las dos clases para obtener los M nuevos individuos de D_{l+1} se obtiene de acuerdo con la siguiente proporción: para la clase $c = 0$ se instancian un total de $\frac{\bar{F_H^l}}{\bar{F_H^l}+\bar{F_L^l}} \cdot M$ individuos, y los otros $\frac{\bar{F_L^l}}{\bar{F_H^l}+\bar{F_L^l}} \cdot M$ se instanciarán para la clase $c = 1$, donde $\bar{F_H^l}$ es la media de los valores obtenidos al evaluar los individuos de la clase de los mejores, mientras que $\bar{F_L^l}$ es la media de los valores obtenidos por los individuos de la clase de los peores. Tanto $\bar{F_H^l}$ como $\bar{F_L^l}$ se calculan suponiendo que todas los individuos tienen un valor positivo, de lo contrario tendría que ser normalizado en consecuencia. Esto es así en los casos en los que se quiere maximizar la función de evaluación $F(\boldsymbol{x})$. En caso de minimizar se instancian un total de $(1 - \frac{\bar{F_H^l}}{\bar{F_H^l}+\bar{F_L^l}}) \cdot M$ individuos para la clase $c = 0$ y $(1 - \frac{\bar{F_L^l}}{\bar{F_H^l}+\bar{F_L^l}}) \cdot M$ para la clase $c = 1$. Igual que en el caso anterior, se supone que todos los individuos tienen un valor positivo.

Utilizando este método, generamos M nuevos individuos de manera que cuanto

mayor sea la diferencia entre $\bar{F}_H^l$ y $\bar{F}_L^l$ más individuos generaremos de la mejor clase ($c = 0$). El propósito de este mecanismo es que el número de individuos generado por la clase de los mejores se adapte, a fin de evitar una convergencia demasiado rápida, tratando de reducir al mínimo el riesgo de caer en óptimos locales.

Existen dos maneras razonables de realizar el paso de selección de las clases para la transición de D_l^K a D_l^C:

- Ignorando la clase media y manteniendo para D_l^C sólo la clase superior e inferior. Esta elección la representaremos como CS_{1+3}.
- Ignorando la clase inferior y manteniendo para D_l^C las dos mejores clases, que lo representaremos por CS_{1+2}.

La otra opción posible (es decir, ignorar la clase de los individuos más aptos, CS_{2+3}) no asegura que la nueva generación de individuos contenga al menos al mejor de los anteriores.

El paso de selección de clases es un parámetro importante en los EBCOAs ya que determina el éxito del proceso de búsqueda debido a su relación directa con la fase de aprendizaje. Sin embargo, otro parámetro importante que también influye directamente tanto en el aprendizaje como en la simulación es la fase de combinación de las poblaciones en la evolución de una generación a la siguiente. Esto se representa en la figura 4.8 como la transición de D_{l+1} a D_{l+1}^K.

Como se ha explicado anteriormente, en la fase de evolución de la población de una generación a la siguiente, se combinan los R individuos de D_l^K con los M nuevos individuos generados recientemente en D_{l+1}. Debemos asegurar que el número de individuos de la l-ésima generación permanecerá en la generación $l + 1$. Esta práctica va más allá del puro enfoque elitista de los algoritmos genéticos (así como de algunos EDAs), de mantener el mejor individuo de la generación anterior, ya que en los EBCOAs también es necesario en algunos casos mantener las características de los individuos peores. Por lo tanto, podemos elegir entre, al menos, tres opciones para combinar los individuos de D_l^K y D_{l+1} y eliminar los restantes, con el fin de crear la población D_{l+1}^C:

1. **Considerar sólo las mejores y las peores clases de generaciones anteriores:** Esta es la opción que representa la figura 4.11a. Consiste en seleccionar los individuos de las mejores y las peores clases de la unión de D_l^K y D_{l+1} y eliminar el resto (los centrales). De esta manera, en la etapa de aprendizaje de EBCOAs, se utilizan los individuos más diferentes encontrados durante todo el proceso de búsqueda, en todas las generaciones anteriores. Esta opción es razonable aplicarla con la selección de clases CS_{1+3} de D_l^K (los mejores y peores individuos). Al hacerlo, mantenemos en D_{l+1}^K, los mejores y peores individuos encontrados durante todas las generaciones.

Esta opción tiene la característica de mantener, en todas las generaciones, tanto a los mejores como a los peores individuos encontrados en cualquiera de las generaciones anteriores. Mantener los mejores individuos resulta apropiado en todos los casos, pero con los peores se crearía el inconveniente, para algunos EBCOAs, de confundir la etapa de aprendizaje ya que en la mayoría de los problemas complejos de optimización, los peores individuos del espacio de búsqueda, son evaluados por lo general en las primeras generaciones. Esto hace que los EBCOAs se estanquen en regiones del espacio de búsqueda evitando que el proceso de búsqueda avance.

2. **Mantener sólo los individuos peores de la última generación:** La forma en que se combinan las poblaciones para esta opción se representa en la figura 4.11b. Esta opción es similar a la anterior, salvo que en este caso no se tienen en cuenta los $R/6$ peores individuos de la población de la generación anterior D_l^K de cara a crear la nueva población D_{l+1}^K. Esta eliminación asegura que la clase de los peores individuos en D_{l+1}^K contendrá un conjunto de individuos compuesto por los peores individuos recién generados en D_{l+1}, así como la mitad de los individuos de la clase de los peores de la generación anterior D_l^K. Esto añade el concepto de *envejecimiento* de la clase de los peores individuos, manteniendo al mismo tiempo esta clase en las sucesivas generaciones, con la esperanza de que los EBCOAs sean capaces de concentrarse más en determinadas regiones del espacio de búsqueda consiguiendo que el proceso avance.

3. **Combinación elitista:** Esta opción selecciona los individuos de las mejores clases de la unión de D_l^K y D_{l+1} y elimina el resto (los peores). Las dos opciones anteriores son más apropiadas con un tipo de selección de clases CS_{1+3} para la transición de D_l^k a D_l^C. Sin embargo, en algunos problemas concretos de optimización es probable que mejore la convergencia si mantenemos los individuos de las dos clases principales en D_l^C. Esta opción se ilustra en la figura 4.11c.

4.3.4 Variables no presentes en el modelo gráfico probabilístico

Otro aspecto importante digno de comentar con respecto a la generación de los nuevos individuos de la población siguiente D_{l+1} es la decisión de cómo instanciar algunas de las variables que no están presentes en el clasificador Bayesiano.

Dado que en los EBCOAs es posible emplear clasificadores Bayesianos que pueden omitir algunas variables predictoras en el modelo gráfico probabilístico (por ejemplo, utilizando el clasificador selective naïve Bayes o semi-naïve Bayes como se explica en la sección 4.2), debemos abordar la cuestión de cómo instanciar estas variables, a fin de generar los valores de las variables correspondientes en los M nuevos individuos. La forma de instanciar variables que no están presentes en el modelo gráfico probabilístico es una situación completamente diferente a que estas variables estén presentes pero

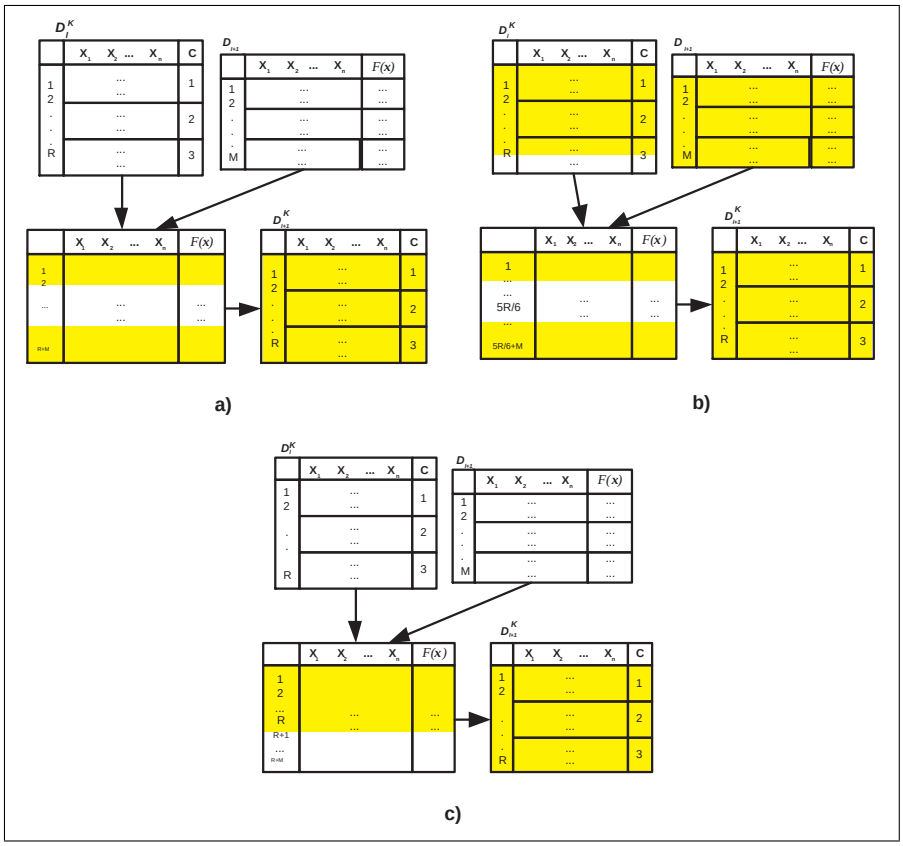

Figura 4.11: Ejemplos de posibles alternativas para el paso de combinación de las poblaciones D_l^K y D_{l+1} en los EBCOAs. Las regiones sombreadas representan individuos que estarán presentes en D_{l+1}^K. El resto se eliminan. En las opciones (a) y (c) se tienen en cuenta todos los individuos de D_l^K y D_{l+1}, mientras que en la opción (b) se eliminan los $R/6$ peores individuos de D_l^K desde el principio, antes de la combinarlos con D_{l+1}.

no conectadas en el modelo gráfico probabilístico ya que, incluso cuando las variables no aparecen conectadas, tienen una distribución de probabilidad $p_l(\boldsymbol{x})$ estimada que se puede utilizar para simular nuevos individuos.

El hecho de que una variable X_i no esté presente en la estructura final implica que los valores asignados a esa variable en los individuos de todas las $|C|$ clases no son relevantes para distinguirlos. Esto tiene una importante lectura, ya que no significa que el valor asignado a esas variables no es intrascendente para la clasificación de un individuo. Hay que tener en cuenta que el individuo constituye un punto en el espacio de búsqueda para un problema específico, y que todos los valores asignados a las variables suelen ser pertinentes para la obtención de un ajuste individual y para que converja a la solución óptima. Por tanto, es necesario un método para generar nuevos individuos con valores en todas las variables (incluyendo los que no aparecen

en el clasificador Bayesiano). Para solucionar este problema proponemos distinguir las dos razones diferentes que hacen que una variable no esté presente en la estructura: por un lado, variables que tienen en todas las clases la misma distribución de probabilidad (*variables irrelevantes*), es decir variables que tienen en todas las clases los mismos valores siempre y por otro lado, variables en las cuales todos los valores aparecen de igual manera en las diversas clases y por lo tanto no reflejan ninguna diferencia entre las características de las clases (*variables redundantes*). En el primer caso, se ha considerado que la estimación de la probabilidad de una variable irrelevante X_i para tomar su k-ésimo valor se calcula como $p(x_i) = p(x_i^k|c)$. Para las variables redundantes, pueden considerarse diferentes opciones a la hora de estimar su probabilidad: se puede considerar que estas variables siguen una distribución de probabilidad uniforme o se puede utilizar la probabilidad con la que aparece dicha variable en los individuos de la clase de los mejores. Proponemos considerar que la distribución de probabilidad, para estas variables redundantes, es uniforme.

El criterio de parada

Todos los pasos anteriores se repiten en EBCOAs hasta que se cumple una condición que parada. Ejemplos de estas condiciones de parada son: realizar un número fijo de generaciones o haber evaluado a un número fijo de individuos diferentes, alcanzar uniformidad en la población generada, o no obtener ningún individuo con un valor mejor de la función objetivo después de cierto número de generaciones.

4.4 Experimentos realizados y resultados obtenidos

En este apartado se describen los experimentos realizados con el fin de probar el rendimiento de los EBCOAs en comparación a otros algoritmos evolutivos como pueden ser los EDAs y los GAs. Estos experimentos se han llevado a cabo utilizando funciones de optimización estándar tanto en dominios discretos como en continuos.

4.4.1 Experimentos sobre problemas de optimización estándar en dominios discretos

Para los experimentos en el dominio discreto hemos tratado de optimizar tres funciones de optimización estándar como son las funciones HIFF, IsoPeak, y IsoTorus, que se sabe que son funciones complejas y con óptimos locales (Santana, 2005). A continuación describimos brevemente estas tres funciones:

1. **HIFF:** La función objetivo para este problema se define como:

$$HIFF(\boldsymbol{x}) = f(x_1, \ldots, x_n)$$

$$HIFF(\boldsymbol{x}) = \begin{cases} 1, & \text{Si } (|s| = 1) \\ |s| + f(x_1, \ldots, x_{\frac{s}{2}}) & \text{Si } (|s| > 1) \\ +f(x_{\frac{s}{2}+1}, \ldots, x_s) & \text{y } (\sum_{i=1}^{|s|} x_i = 0), \\ & \text{o } (\sum_{i=1}^{|s|} x_i = |s|) \\ f(x_1, \ldots, x_{\frac{s}{2}}) & \\ +f(x_{\frac{s}{2}+1}, \ldots, x_s) & \text{en otro caso} \end{cases} \tag{4.11}$$

El óptimo se obtiene cuando todas las variables del individuo toman el valor 0 o el valor 1, esto es $(0, 0, \ldots, 0)$ ó $(1, 1, \ldots, 1)$. Con un tamaño de individuo $n = 64$, el valor óptimo es $Opt = 448$.

2. **IsoPeak:** En este caso, la función objetivo se define como:

$F_{IsoPeak}(\boldsymbol{x}) = IsoC_2(x_1, x_2) + \sum_{i=2}^{m} IsoC_1(x_i, x_{i+1})$

siendo

x	00	01	10	11
$IsoC_1$	m	0	0	$m-1$
$IsoC_2$	0	0	0	m

y $n = m+1$. El óptimo se obtiene cuando todas las variables del individuo valen 1, esto es $(1, 1, \ldots, 1)$ con un valor de $m \cdot (m-1)+1$. Con un tamaño de individuo $n = 64$, el óptimo es $Opt = 3907$.

3. **IsoTorus:** La función objetivo para este problema se define como:

$F_{IsoTorus} = IsoT_1(x_{1-m+n} + x_{1-m+n} + x_1 + x_2 + x_{1+m}) + \sum_{i=2}^{n} IsoT_2(x_{up} + x_{left} + x_i + x_{right} + x_{down})$

donde $x_{up}, x_{left}, x_i, x_{right}, x_{down}$ se definen como los vecinos correspondientes y

$$IsoT_1 = \begin{cases} m & \text{Si } u = 0 \\ m-1 & \text{Si } u = 5 \\ 0 & \text{en otro caso} \end{cases}$$
$$IsoT_2 = \begin{cases} m^2 & \text{Si } u = 5 \\ 0 & \text{en otro caso} \end{cases} \tag{4.12}$$

siendo $n = m^2$. El óptimo se obtiene también cuando todas las variables del individuo toman el valor 1, esto es $(1, 1, \ldots, 1)$. Con un tamaño de individuo $n = 64$, el valor óptimo es $Opt = 505$.

Queremos comparar la eficiencia de los EBCOAS en relación con los EDAs y los algoritmos genéticos (GAs). Para ello, a la hora de realizar los experimentos, además de probarlos con los EBCOAs utilizando los diferentes clasificadores descritos en la sección 4.2: *naïve Bayes*, *selective naïve Bayes*, *semi–naïve Bayes* y *tree augmented naïve Bayes*, hemos elegido EDAs que tienen en cuenta distinto número de dependencias entre las variables, concretamente *UMDA* (Mühlenbein, 1998), *MIMIC* (de Bonet y col., 1997), y $EBNA_{BIC}$ (Etxeberria y Larrañaga, 1999). Como GAs ampliamente conocidos hemos elegido el básico (cGA) (Holland, 1975), el *elitist* (eGA) (Whitley y Kauth, 1988) y el *steady state* (ssGA) (Michalewicz, 1992).

En los experimentos realizados con EBCOAs, hemos dividido la población en 3 clases diferentes ($|K| = 3$) de las cuales, para la fase de aprendizaje seleccionamos a los mejores y peores individuos ($|C| = 2$). Esto se ilustra en la Figura 4.12.

D_l^K

	X_1	X_2	X_3	...	X_n	K
1	4	1	5	...	3	H
2	2	3	4	...	6	H
...	...	...	...	...	...	...
R/3	3	1	4	...	2	H
R/3 +1	2	3	1	...	6	M
...	...	...	...	...	...	...
2R/3	1	5	4	...	2	M
2R/3 +1	4	2	6	...	6	L
...	...	...	...	...	...	...
R	1	5	7	...	1	L

Figura 4.12: En nuestros experimentos dividimos D_l^K en tres clases (H, M y L), de las que cuales se seleccionan dos (H y L) para el aprendizaje. La clase M se ignora para esta etapa.

El criterio de parada en todos los experimentos es obtener la solución óptima o llegar a la generación número 500. Se han realizado 10 ejecuciones para cada función con cada uno de los algoritmos utilizados. La tabla 4.5 muestra la media del

valor obtenido por el mejor individuo en la última generación, así como el número de generaciones realizadas para llegar a la solución final, para cada uno de los experimentos. Las primeras líneas de la tabla se refieren a los resultados obtenidos al ejecutar los EBCOAs utilizando diferentes clasificadores: naïve Bayes ($EBCOA_{NB}$), selective naïve Bayes ($EBCOA_{SNB}$), semi–Naïve Bayes con selección secuencial hacia delante y unión ($EBCOA_{FSSJ}$), semi–naïve Bayes con eliminación secuencial hacia atrás y unión ($EBCOA_{BSEJ}$) y tree augmented naïve Bayes ($EBCOA_{TAN}$). En la misma tabla 4.5 se muestran los resultados obtenidos utilizando diferentes EDAs según el número de dependencias entre variables que consideran: UMDA (EDA_{UMDA}), MIMIC (EDA_{MIMIC}) y EBNA$_{BIC}$ (EDA_{EBNA}) y por último los resultados al ejecutar los GAs: básico (cGA), el elitist (eGA) y el steady state ($ssGA$). Las columnas *Val* y *Ev* representan los valores medios de 10 ejecuciones del mejor valor obtenido para la función objetivo en la última generación y el número de evaluaciones realizadas al terminar la ejecución respectivamente.

Recordemos que para la función HIFF, con un tamaño de individuo de $n = 64$, el óptimo es el valor 448. Podemos encontrar, en cada uno de los grupos de algoritmos, EBCOAs, EDAs y GAs, algún algoritmo que alcanza el óptimo en todas las ejecuciones, concretamente $EBCOA_{TAN}$, EDA_{EBNA} y ssGA. Al aplicar el test de hipótesis de Kruskal- Wallis (Kruskal y Wallis, 1952) al conjunto de valores obtenidos al ejecutar los diferentes algoritmos, se encuentran diferencias estadísticamente significativas entre los algoritmos que siempre encuentran el óptimo y el resto de EDAs y EBCOAs, excepto con $EBCOA_{SNB}$, con el que no encuentra diferencias estadísticamente significativas. Tampoco se encuentran diferencias estadísticamente significativas entre los diferentes genéticos utilizados.

La función IsoPeak tiene un óptimo local (3906) que corresponde a individuos con todas las variables a cero, muy cercano al óptimo global (3907). Este hecho confunde a la mayoría de los algoritmos, y aunque alguno de ellos es capaz de encontrar el óptimo a veces (3 veces el EDA_{EBNA} y 1 vez el $ssGA$), sólo el $EBCOA_{TAN}$ es capaz de encontrar el óptimo en las 10 ejecuciones. Si aplicamos el test de hipótesis de Kruskal- Wallis a los valores obtenidos en esta función, observamos que se encuentran diferencias significativas entre $EBCOA_{TAN}$ y el resto de algoritmos utilizados, excepto con EDA_{EBNA} y $ssGA$.

La función IsoTorus también tiene óptimos locales, y tanto los EDAs como los GAs caen en ellos en algunas de las ejecuciones. De las 10 ejecuciones de cada algoritmo, la mayoría de los EDAs y GAs fueron capaces de encontrar el óptimo global algunas veces (EDA_{MIMIC} una vez, EDA_{EBNA} y cGA 4 veces, y $ssGA$ y eGA 5 veces). Sin embargo, $EBCOA_{NB}$ y $EBCOA_{TAN}$ encuentran el óptimo global en las 10 ejecuciones, mientras que $EBCOA_{BSSJ}$ y $EBCOA_{SNB}$ lo encuentran 8 y 2 veces respectivamente. Al aplicar el test de hipótesis de Kruskal- Wallis y comparar los dos algoritmos que encuentran siempre el óptimo, $EBCOA_{NB}$ y $EBCOA_{TAN}$, con el resto de algoritmos, encontramos diferencias estadísticamente significativas con $EBCOA_{SNB}$, $EBCOA_{BSEJ}$, EDA_{UMDA} y EDA_{MIMIC}.

	HIFF		IsoPeak		IsoTorus	
	Ev.	Val.	Ev.	Val.	Ev.	Val.
$EBCOA_{NB}$	105036.8	290	51995.4	3906	25175.9	505
$EBCOA_{SNB}$	94640.7	355.2	43910.0	3906	207914.1	472
$EBCOA_{FSSJ}$	249838.2	290.2	249893.5	3859.8	227610.3	471.6
$EBCOA_{BSEJ}$	189178.9	184.5	58694.3	3803.8	66701.9	474.3
$EBCOA_{TAN}$	4589.9	448	4391.8	3907	3989.6	505
EDA_{UMDA}	107120.4	295.6	67303.3	3905.5	47244.7	400.3
EDA_{MIMIC}	97572.0	283.2	69385.9	3906	46941	422.3
EDA_{EBNA}	23336.0	448	19708.6	3906.3	28703.0	485.2
cGA	202000	395.2	202000	3628.1	202000	477.2
eGA	202000	388.8	202000	3793.7	202000	488.5
$ssGA$	202000	448	202000	3906.1	202000	488.5

Tabla 4.1: Media de los resultados obtenidos en 10 ejecuciones con cada algoritmo y función objetivo. Las columnas *Val* y *Ev* representan el mejor valor obtenido para la función objetivo en la última generación y el número de evaluaciones realizadas respectivamente.

Todo ello nos lleva a concluir que el comportamiento de los EBCOAs con el clasificador TAN es estadísticamente comparable e incluso mejor al de EDAs y GAs. En los casos en que se utilizan otros clasificadores, podemos considerar su comportamiento estadísticamente comparables al de EDAs y GAs, aunque varía dependiendo del problema concreto.

4.4.2 Experimentos sobre problemas de optimización estándar en dominios continuos

También nos interesa conocer el rendimiento de los EBCOAs en entornos continuos en comparación con los EDAs continuos y Estrategias Evolutivas (ES) (Schwefel, 1995). A continuación se describen los experimentos realizados y los resultados obtenidos con este propósito.

Para la comparación, hemos elegido EDAs continuos que tienen en cuenta diferente número de dependencias entre las variables: UMDA$_c$, MIMIC$_c$, EGNA$_{ee}$ y EGNA$_{BGe}$. Estos algoritmos se han comentado en la sección 3.3. Con respecto a ES, elegimos CMA-ES (Evolutionary Strategy with Covariance matrix Adaptation) (Hansen y Kern, 2004; Hansen y col., 2003) ya que se considera uno de los algoritmos con un mejor rendimiento en problemas de optimización en dominios continuos.[2]

Los problemas de optimización seleccionadas son los propuestos por (Bengoetxea

[2]En este caso se ha utilizado el programa de MATLAB cmaes.m versión 2.34 que se encuentra disponible en la web en `http://www.icos.ethz.ch/software/evolutionary_computation/cmaes.m`

y col., 2001b) para comparar los algoritmos de computación evolutiva en el dominio continuo: *Ackley* (Ackley, 1987; Bäck, 1996), *Griewangk* (Törn y Zilinskas, 1989), *Rosenbrock generalized* (Rosenbrock, 1960; Salomon, 1998), *Sphere Model* y *Summation Cancelation* (Baluja y Davies, 1997).

Estos problemas de optimización se describen brevemente a continuación.

1. **Ackley:** La función objetivo consiste en minimizar la siguiente función:

$$\begin{aligned} F(\boldsymbol{x}) &= -20 \cdot \exp\left(-0{,}2\sqrt{\frac{1}{n}\cdot\sum_{i=1}^{n} x_i^2}\right) \\ &- \exp\left(\frac{1}{n}\cdot\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1). \end{aligned} \tag{4.13}$$

   Por lo tanto, el valor óptimo es 0, que se obtiene cuando el individuo tiene todas sus variables a 0. El problema se define con $-20 \leq x_i \leq 30$, $i = 1, \ldots, n$.

2. **Griewangk:** Es también un problema de minimización definido como:

$$F(\boldsymbol{x}) = 1 + \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{4.14}$$

   donde el rango de todos los componentes del individuo es $-600 \leq x_i \leq 600$, $i = 1, \ldots, n$, y el valor óptimo es 0, que igual que en el caso anterior, se obtiene cuando todos los componentes del individuo son 0.

3. **Rosenbrock generalizado.** Se trata también de un problema de minimización definido como:

$$F(\boldsymbol{x}) = \sum_{i=1}^{n-1}\left[\, 100\cdot(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \,\right]. \tag{4.15}$$

   Una vez más, el valor óptimo es 0 y se obtiene cuando todas las variables del individuo toman el valor 1. El rango de valores de las variables es $-10 \leq x_i \leq 10$, $i = 1, \ldots, n$.

4. **Sphere model:** Es un sencillo problema de minimización, cuya función objetivo es:

$$F(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2. \tag{4.16}$$

El rango de los componentes del individuo se define como $-600 \leq x_i \leq 600$, $i = 1, \ldots, n$. El óptimo es cero y se obtiene cuando todas las variables del individuo toman el valor 0.

5. **Summation cancelation.** En este caso, el rango de valores para las variables de un individuo es $-0{,}16 \leq x_i \leq 0{,}16$, $i = 1, \ldots, n$, y la función objetivo se calcula:

$$F(\boldsymbol{x}) = \frac{1}{(10^{-5} + \sum_{i=1}^{n} |y_i|)} \tag{4.17}$$

donde $y_1 = x_1$, $y_i = x_i + y_{i-1}$, $i = 2, \ldots, n$. El óptimo se obtiene cuando todas las variables del individuo valen 0. En ese caso, el valor de la función es 100000.

Para estos experimentos hemos elegido un tamaño de población de 400 ($R = 400$), ya que consideramos que es un tamaño razonable para que los EBCOAs puedan distinguir las características entre los mejores y peores individuos. Hemos utilizado individuos de 10 variables ($n = 10$). En el caso de los EBCOAs, asumimos $|K| = 3$ y $|C| = 2$ (la clase de los mejores y la de los peores). El tamaño que utilizamos para las clases mejores y peores es de $R/4$, siendo R el tamaño de la población. En los EDAs continuos realizamos una selección del subconjunto de los $R/2$ mejores para estimar la función de densidad, y una aproximación elitista. Finalmente, para CMA-ES utilizamos el mismo tamaño de población que en los casos anteriores ($R = 400$) y 200 como número máximo de padres, mientras que para el resto de los parámetros se han dejado los valores por defecto que sugieren los autores (Hansen y Kern, 2004; Hansen y col., 2003).

El criterio de parada establecido en todos los casos consiste en encontrar la solución óptima (asumiendo un margen de error de 10^{-6} del óptimo global), o alcanzar un máximo de 150 generaciones o no obtener ninguna mejora en la población de la última generación. Cada algoritmo se ha ejecutado 10 veces con cada uno de los problemas de optimización comentados.

La tabla 4.2 muestra la media de los valores obtenidos en las 10 ejecuciones, representando el valor obtenido por el mejor individuo (V) y el número de evaluaciones realizadas (E). Estos resultados muestran que los EBCOAs obtienen resultados similares a los EDAs aunque necesitan mayor número de evaluaciones. Sin embargo, CMA-ES obtiene mejores resultados.

Al aplicar el test de hipótesis de Kruskal- Wallis (Kruskal y Wallis, 1952) para comprobar si existen diferencias significativas en el comportamiento de los diferentes algoritmos, comprobamos que, en aquellos que no tienen óptimos locales (como

	Ackley		Griewangk		Rosenbrock	
$n = 10$	V	E	V	E	V	E
EBCOA$_{NB}$	8.9E-6	46365	1.3E-1	60250	8.0E+0	60250
EBCOA$_{S_{NB}}$	8.8E-6	44689	6.7E-5	39662	8.6E+0	40899
EBCOA$_{TAN}$	8.9E-6	54185	3.3E-1	60250	7.9E+0	60250
UMDA$_c$	8.8E-6	23063	5.4E-2	58814	8.7E+0	52589
MIMIC$_c$	7.8E-6	23382	8.2E-2	59093	8.7E+0	44968
EGNA$_{ee}$	7.9E-6	22983	5.4E-2	58654	8.6E+0	28889
EGNA$_{BGe}$	8.5E-6	22904	9.2E-2	54784	8.6E+0	26375
CMA-ES	1.9E-7	23962	2.7E-8	14562	3.9E-8	44082

	Sphere		SumCan	
$n = 10$	V	E	V	E
EBCOA$_{NB}$	6.5E-6	38704	1.6E+03	60250
EBCOA$_{S_{NB}}$	7.6E-6	38704	1.6E+04	60250
EBCOA$_{TAN}$	7.5E-6	45088	8.9E+01	60250
UMDA$_c$	7.3E-6	15003	1.6E+04	60250
MIMIC$_c$	6.7E-6	15163	1.7E+04	60250
EGNA$_{ee}$	6.7E-6	14884	1.0E+05	37108
EGNA$_{BGe}$	7.0E-6	14884	1.0E+05	37826
CMA-ES	3.7E-8	13802	1.0E+05	45682

Tabla 4.2: Media de los resultados obtenidos en 10 ejecuciones con cada algoritmo y función objetivo utilizando un tamaño de población de $R = 400$. Las columnas V y E representan el mejor valor obtenido y el número de evaluaciones realizadas respectivamente.

el Sphere) o en los que tienen pequeños óptimos locales (como Ackley), CMA-ES obtiene diferencias significativas con el resto de algoritmos (excepto con MIMIC$_c$ y EGNA$_{ee}$ para la función Ackley). Sin embargo, no se encuentran diferencias significativas entre EBCOAs y EDAs. Con la función Griewangk, no se encuentran diferencias significativas entre CMA-ES y EBCOA$_{S_{NB}}$, aunque sí entre CMA-ES y el resto de algoritmos. En aquellos problemas de optimización con importantes óptimos locales (como Summation Cancelación), el mejor comportamiento lo encontramos en los algoritmos CMA-ES, EGNA$_{ee}$ y EGNA$_{BGe}$, entre los que no se encuentran diferencias significativas, aunque sí se encuentran entre estos tres algoritmos y el resto.

La función Rosenbrock es especialmente difícil de optimizar debido a que el óptimo global se encuentra en medio de una región bastante plana. Esta es la razón de que los EDAs no encuentren el óptimo global, cayendo en óptimos locales. Los EBCOAs tampoco consiguen mejorar estos resultados y funcionan de manera similar a los EDAs. CMA-ES si es capaz de obtener el óptimo global. Sin embargo, al aplicar el test de hipótesis a los datos obtenidos en la función Rosenbrock, no se encuentran diferencias significativas entre el comportamiento de CMA-ES y el de EBCOA$_{NB}$ y EBCOA$_{TAN}$,

	Ackley		Griewangk		Rosenbrock	
$n = 10$	V	E	V	E	V	E
EBCOA$_{NB}$	9.1E-6	18329	1.6E-2	35841	7.9E+0	23125
EBCOA$_{S_{NB}}$	8.3E-6	17692	8.7E-6	19165	7.8E+0	14031
EBCOA$_{TAN}$	8.7E-6	22229	3.6E-1	26329	7.5E+0	59900

	Sphere		SumCan	
$n = 10$	V	E	V	E
EBCOA$_{NB}$	7.9E-6	16419	1.6E+04	59900
EBCOA$_{S_{NB}}$	6.1E-6	15921	8.1E+04	48557
EBCOA$_{TAN}$	7.4E-6	19603	2.6E+04	59900

Tabla 4.3: Media de los resultados obtenidos en 10 ejecuciones con un tamaño de población de $R = 200$. Las columnas V y E representan el mejor valor obtenido y el número de evaluaciones realizadas respectivamente.

y sí con el resto.

Hemos repetido estos experimentos con los EBCOAs modificando el tamaño de la población para comprobar si esto influye en el comportamiento de estos algoritmos. Ahora utilizamos un tamaño de población de 200 ($R = 200$). También modificamos el criterio de parada utilizando un máximo de 300 generaciones de manera que se mantenga el número total de individuos evaluados. Para el resto de parámetros mantenemos los mismos criterios que en el caso anterior. Los resultados obtenidos al ejecutar 10 veces estos nuevos experimentos se muestran en la tabla 4.3.

Como podemos comprobar, al disminuir el tamaño de población mejoran los resultados anteriores en aquellas funciones en las que se consigue alcanzar el óptimo. En general, se reduce el número de evaluaciones necesarias. Esto nos lleva a considerar que en los EBCOAs es importante exigir algunas condiciones concretas para asegurar un buen rendimiento, como por ejemplo el uso de una población de individuos de un tamaño adecuado, así como una buena elección en la selección de clases y combinación de la población.

A continuación nos proponemos realizar un análisis de los parámetros que influyen en el rendimiento de estos algoritmos y la relación que puede haber entre ellos.

4.4.3 Análisis de la influencia de determinados parámetros en el rendimiento de los EBCOAs

Con el fin de comprobar el comportamiento de EBCOAs ante parámetros como el tamaño de la población, la elección en la selección de las clases y combinación de la población, hemos realizado experimentos con las diferentes opciones comentadas en la sección 4.3.3, sobre tres problemas de optimización presentados anteriormente: Ackley, Griewangk y Sphere. La tabla 4.4 muestra los resultados obtenidos al ejecutar

los EBCOAs con el clasificador naïve Bayes (EBCOA$_{NB}$) y con TAN (EBCOA$_{TAN}$) para cada alternativa de selección de clases y combinación de poblaciones. Hemos comprobado que el tamaño de población desempeña un papel importante en el rendimiento general de EBCOAs, obteniendo diferente comportamiento dependiendo del clasificador utilizado. En este caso se ha utilizado un tamaño de población de 200 para el caso de naïve Bayes y de 25 para TAN. Estos tamaños se han elegido después de comprobar que son los que mejores resultados obtienen, de entre los probados, en cada caso. Esto se ha hecho para proporcionar una comparación con independencia de los diferentes tamaños de la población. Los resultados mostrados en la tabla son la media de 30 ejecuciones en cada uno de los casos y muestran el porcentaje de casos en que el algoritmos encuentra el óptimo (columna *Cv.*) y el número de evaluaciones realizada (columna E). Como criterio de parada se ha utilizado el hecho de que el algoritmo converja permitiendo un máximo de 60000 evaluaciones.

Los resultados obtenidos son muy diferentes para los dos tipos de EBCOAs, y esto ha de entenderse teniendo en cuenta las diferencias en la fase de aprendizaje de estos dos algoritmos. Si consideramos el caso de EBCOAs en los que no hay aprendizaje (es decir, EBCOA$_{NB}$ tiene una estructura fija), el comportamiento es similar al de un UMDA continuo (UMDA$_c$) si la proporción de $\frac{\bar{F}_H^l}{\bar{F}_H^l + \bar{F}_H^l}$ está muy cerca de 1. Esta proporción se eleva a un máximo proporcionando una convergencia más rápida cuando elegimos la primera opción presentada en la sección 4.3.3 (las mejores y peores clases de las generaciones anteriores) para combinar la población y CS$_{1+2}$. Sin embargo, esta velocidad de convergencia es muy apropiada en problemas como el modelo Sphere y Ackley en el que los óptimos locales son inexistentes o pequeños, pero cuando los óptimos locales son más profundos, como en el problema Griewangk, el algoritmo funciona mejor cuando las opciones de selección de clases y combinación de poblaciones son CS$_{1+2}$ y elitista. La razón de esto último es, probablemente, que la complejidad del problema recomienda concentrarse en los mejores individuos, con la ayuda de los individuos intermedios, con el fin de proporcionar un medio de escapar de estos óptimos locales buscando en las regiones cercanas a las que contienen a los individuos más aptos. Es diferente en el caso de otros EBCOAs que consideran más dependencias o procedimientos más complejos en la fase de aprendizaje. Este es el caso de EBCOA$_{TAN}$, para el que la fase de aprendizaje necesita construir una nueva estructura en cada generación. Por lo tanto, diferentes opciones de selección de clases y combinación de la población darían lugar a diferentes tipos de convergencia. La naturaleza y la complejidad del problema de optimización también determina el mejor tipo de parámetros que se debe aplicar en EBCOAs.

Como muestra la Tabla 4.4, para el tipo de problemas elegidos para nuestros experimentos, la mejor opción para EBCOA$_{NB}$ es una selección de clase CS$_{1+2}$ con una combinación elitista. Por otra parte, en EBCOA$_{TAN}$ la mejor opción es una combinación de la población de las mejores y peores clases de las generaciones anteriores y una selección de clases CS$_{1+3}$.

	Ackley				Griewangk			
EBCOA$_{NB}$	CS$_{1+2}$		CS$_{1+3}$		CS$_{1+2}$		CS$_{1+3}$	
Combinación Pob.	Cv.	E	Cv.	E	Cv.	E	Cv.	E
Mejores/peores clases	93	45615	83	35749	90	34030	80	28201
Sólo peores recientes	83	44099	77	39221	90	33057	83	30090
Combinación elitista	93	40746	80	37886	93	30690	70	28818
EBCOA$_{TAN}$	CS$_{1+2}$		CS$_{1+3}$		CS$_{1+2}$		CS$_{1+3}$	
Combinación Pob.	Cv.	E	Cv.	E	Cv.	E	Cv.	E
Mejores/peores clases	100	56319	100	30375	87	55907	93	28248
Sólo peores recientes	100	55000	100	32651	80	56159	97	24419
Combinación elitista	100	40564	100	31688	100	39174	93	29003

	Sphere			
EBCOA$_{NB}$	CS$_{1+2}$		CS$_{1+3}$	
Combinación Pob.	Cv.	E	Cv.	E
Mejores/Peores clases	97	40000	77	32931
Sólo peores recientes	97	38991	63	35276
Combinación elitista	90	35932	90	33588
EBCOA$_{TAN}$	CS$_{1+2}$		CS$_{1+3}$	
Combinación Pob.	Cv.	E	Cv.	E
Mejores/Peores clases	100	47891	100	26567
Sólo peores recientes	100	47599	100	30295
Combinación elitista	100	34805	100	27260

Tabla 4.4: Media de los resultados obtenidos después de 30 ejecuciones con cada EBCOA y diferentes opciones de selección de clases y combinación de las poblaciones D_l^K y D_{l+1} para obtener D_{l+1}^K. El tamaño de la población se ha fijado en 200 y 25 para EBCOA$_{NB}$ y EBCOA$_{TAN}$ respectivamente. La columna *Cv.* representa el porcentaje de convergencia y E el número de evaluaciones realizadas. Se permitió un máximo de 60.000 evaluaciones.

4.4.4 La influencia del tamaño de la población

En las diferentes pruebas realizadas con los EBCOAs hemos podido apreciar que su comportamiento es muy dependiente de la forma en que diferentes individuos de cada clase representan distintas regiones del espacio de búsqueda. El algoritmo se comporta de forma más apropiada cuando el tamaño de la población se ha elegido de acuerdo a la complejidad del problema. Este parámetro también resulta crítico en EDAs y GAs (Yu y col., 2007; Pelikan y col., 2000). El tamaño de la población R se ha establecido de tal manera que la división en clases también representa a los individuos que se agrupan en regiones con similar valor fitness, ya que este parámetro compromete la tasa de convergencia, al menos, tanto como la selección de clases y la combinación de poblaciones. Además, este parámetro es muy dependiente del problema y su elección

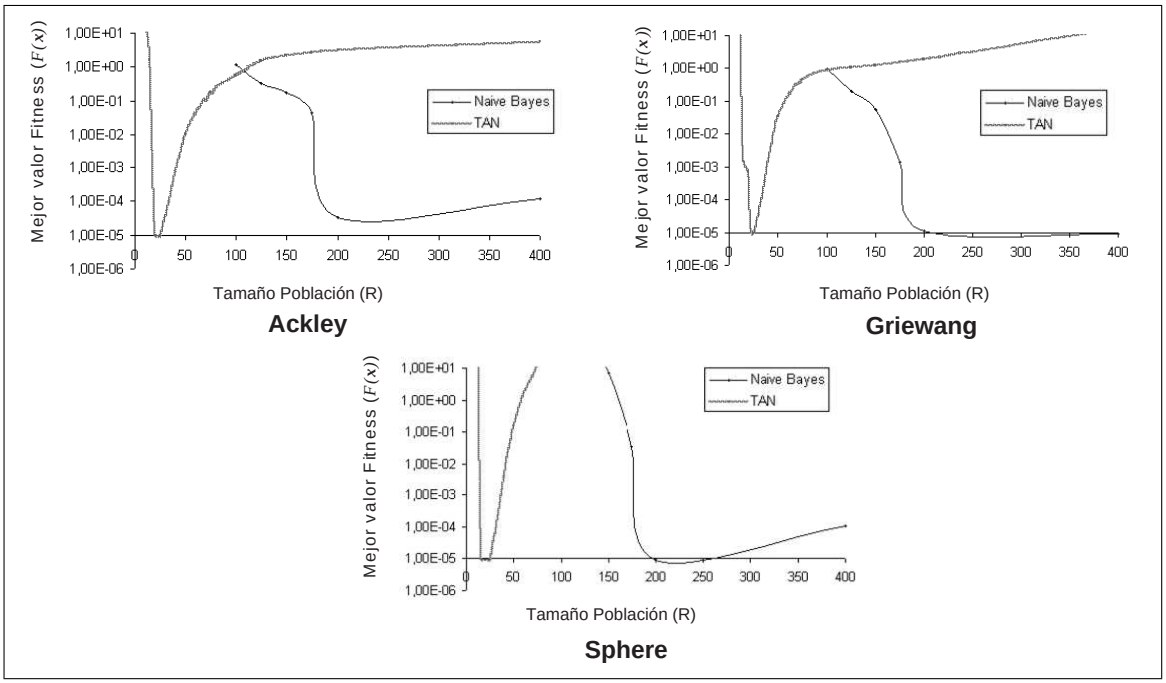

Figura 4.13: Tres ejemplos de cómo afecta el tamaño de la población en el rendimiento general de los EBCOAs. En los tres problemas (Ackley, Griewangk y Sphere con $n = 100$) la gama de tamaños de población que convergen cuando utilizamos EBCOA$_{TAN}$ es menor que los tamaños de población necesarios para alcanzar la convergencia cuando utilizamos EBCOA$_{NB}$. Los experimentos con EBCOA$_{NB}$ y EBCOA$_{TAN}$ fueron realizados utilizando la selección de clases y combinación de poblaciones que mejores valores obtuvo para cada uno de los problemas en los casos que se muestran en la tabla 4.4. El eje x representa el tamaño de la población (R), mientras que el eje y representa el mejor valor fitness obtenido (0 es el óptimo global en los tres problemas de optimización).

tiene que adaptarse a las particularidades de cada uno de los problemas para permitir un adecuado rendimiento en los EBCOAs. Las gráficas de la figura 4.13 reflejan este aspecto para los problemas de optimización Ackley, Griewangk y Sphere. Aquí se muestra que, a fin de obtener un comportamiento comparable, el tamaño de la población para EBCOA$_{TAN}$ debe ser mucho menor que cuando utilizamos EBCOA$_{NB}$. La diferente naturaleza de estos tres problemas de optimización muestra el comportamiento de los EBCOAs cuando no existe un óptimo local (modelo Sphere) o cuando los óptimos locales son más o menos profundos (Griewangk y Ackley respectivamente). Esta figura sugiere que el tamaño de la población influye en el rendimiento global de los EBCOAs, y nuestros experimentos mostraron que su efecto es aún mayor que con la selección de clases o la combinación de poblaciones.

4.4.5 Estudio sobre reajuste de los parámetros de los EBCOAs y sus valores más adecuados

Con el fin de tener un panorama ilustrativo de la importancia relativa de las dependencias a la hora de fijar los parámetros en los EBCOAs, hemos realizado un estudio general en el que hemos ajustado los principales parámetros. Este tipo de estudio sobre las dependencias entre las diferentes configuraciones de parámetros de GAs, para ofrecer el mejor ajuste, ha sido una importante tendencia de investigación desde hace mucho tiempo (Ballester y col., 2005; Barr y col., 1995).

En esta sección presentamos un estudio similar para los EBCOAs, mostrando estas dependencias en forma de una red Bayesiana que representa la relación entre los diferentes parámetros en los tres problemas de optimización definidos previamente. Como hemos mencionado antes, cada combinación diferente de los valores de los parámetros se ha ejecutado un total de 30 veces. Los diferentes parámetros que se combinaron en las pruebas son los siguientes:

- **Problemas de optimización:** Ackley, Griewangk, Sphere model.
- **Tamaño de los individuos del problema:** 50, 100, 150.
- **Tipo de clasificador EBCOA:** EBCOA$_{NB}$, EBCOA$_{TAN}$.
- **Tamaño de la población:** 10, 15, 20, 25, 50, 100, 150, 200, 400.
- **Tipo de selección de clases:** CS$_{1+2}$, CS$_{1+3}$ (ver sección 4.3.3) .
- **Tipos de combinación de poblaciones:** Mejores/peores clases de generaciones anteriores, sólo peores recientes, combinación elitista (ver sección 4.3.3).

Para cada ejecución, hemos recogido los resultados obtenidos en forma de tres variables: si se alcanzó el óptimo antes de realizar un máximo de 60000 evaluaciones ($Cv.$), el número de evaluaciones realizadas (E) y mejor valor obtenido ($F(\boldsymbol{x})$).

Para ilustrar mejor las dependencias e importancia relativa que tienen los diferentes parámetros en el rendimiento de los EBCOAs, hemos creado una red Bayesiana para cada tipo de EBCOA añadiendo estos tres resultados ($Cv.$, E y $F(\boldsymbol{x})$). La figura 4.14 muestra las dos redes Bayesianas obtenidas. Como era de esperar, estas redes Bayesianas muestran que uno de los parámetros más importantes para un buen rendimiento en EBCOAs es el tamaño de la población. Podemos comprobar esta conclusión en la figura 4.13, donde las redes Bayesianas muestran la evidencia de que este parámetro se encuentra entre los más importantes. Además, también es muy importante tener en cuenta las diferencias que presentan EBCOA$_{NB}$ y EBCOA$_{TAN}$ en su fase de aprendizaje al elegir los parámetros de ejecución, ya que los parámetros de selección de clases y combinación de poblaciones no parecen ser muy significativos para EBCOA$_{NB}$, mientras que son críticos para EBCOA$_{TAN}$. Estos resultados también eran de esperar al

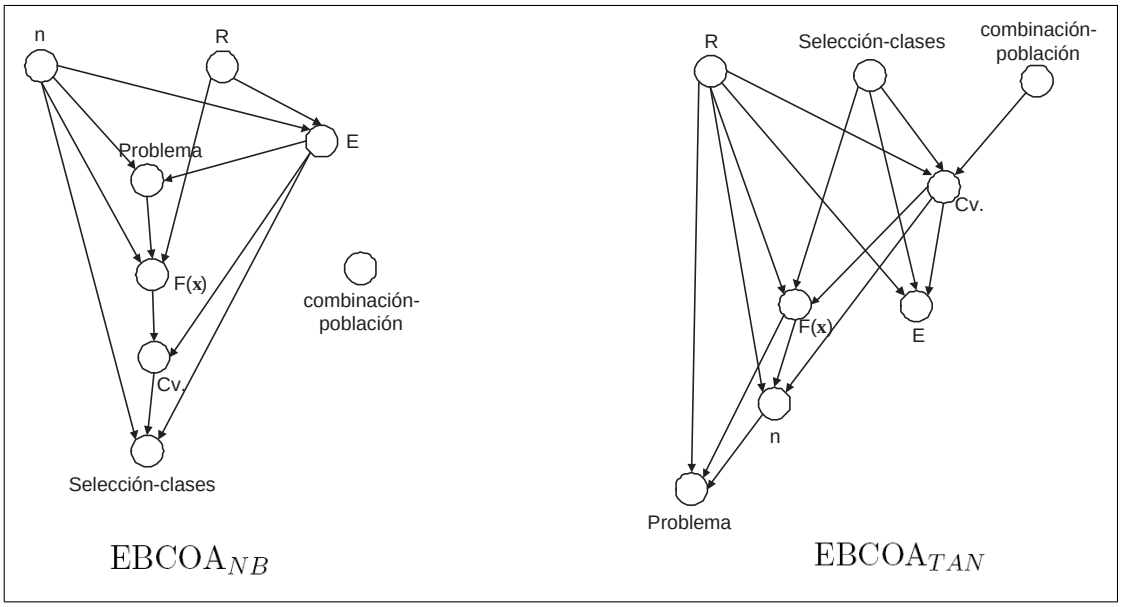

Figura 4.14: Estructuras de redes Bayesianas que ilustran los resultados obtenidos en los experimentos combinando los diferentes parámetros para los EBCOAs. La primera red Bayesiana representa los valores para EBCOA$_{NB}$ y la segunda para EBCOA$_{TAN}$. Los nodos de las redes Bayesianas representan el tamaño de la población (R), tamaño del individuo (n), el problema de optimización (*problema*), el tipo de selección de clases (*selección-clases*), tipo de combinación de poblaciones (*combinación-poblaciones*), la convergencia (*Cv.*), el número de evaluaciones (E) y el mejor valor fitness obtenido ($F(\boldsymbol{x})$).

ver, en el caso de EBCOA$_{TAN}$, que la convergencia del algoritmo mejora o empeora en función de la exactitud del clasificador Bayesiano para representar las características de los mejores o peores individuos, y estos dos parámetros son esenciales para determinar la naturaleza de los individuos que formarán cada una de las clases de D_l^C. En EBCOA$_{NB}$, el hecho de tener una estructura fija, lo hace menos dependiente de estos parámetros.

En lo que respecta a las relaciones entre las tres variables obtenidas (*Cv.*, E y $F(\boldsymbol{x})$ y su relativa dependencia del problema de optimización (*problema*) mostrada en la figura 4.14, hay que señalar que todos estos problemas tienen el mismo valor 0 para el óptimo global, lo que explica por qué los arcos entre estos nodos tienen diferentes direcciones y características en ambas redes Bayesianas.

4.4.6 Comparativa del rendimiento de los EBCOAs con otras técnicas de computación evolutiva en dominios continuos

Una vez analizada la repercusión de los diferentes parámetros, nos interesa conocer el rendimiento de los EBCOAs en entornos continuos, utilizando las conclusiones obte-

		150			200			400	
EBCOA$_{NB}$	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E
Ackley	30	6.0E-2	28,050	90	1.6E-3	40,746	0	1.3E-4	-
Griewangk	40	3.0E-2	26,747	90	2.4E-3	30,690	100	-	56,034
Sphere	30	2.3E-1	30,512	90	9.5E-5	38,991	0	1.2E-4	-
		15			20			25	
EBCOA$_{TAN}$	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E
Ackley	20	2.1E+0	21,706	100	-	23,049	100	-	30,375
Griewangk	80	8.8E-3	13,286	90	9.9E-3	30,265	100	-	39,174
Sphere	100	-	11,603	100	-	15,620	100	-	26,567
		200			400			600	
UMDA$_c$	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E
Ackley	20	1.2E-1	49,950	0	3.7E-3	-	0	1.0E-1	-
Griewangk	100	-	29,692	100	-	59,532	0	4.1E-3	-
Sphere	100	-	35,980	0	4.3E-4	-	0	2.4E-1	-
		200			400			600	
MIMIC$_c$	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E
Ackley	0	5.4E-5	-	0	8.3E-4	-	0	3.3E-2	-
Griewangk	0	2.1E-5	-	30	1.1E-5	51,339	0	5.4E-4	-
Sphere	0	2.2E-5	-	0	3.0E-5	-	0	3.7E-2	-
		400			600			800	
EGNA$_{ee}$	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E
Ackley	0	5.2E+00	-	0	5.1E+00	-	0	6.0E+00	-
Griewangk	0	1.6E-01	-	0	6.4E-03	-	0	2.6E-02	-
Sphere	0	1.3E+01	-	0	2.7E-01	-	0	1.7E+00	-
		25			50			100	
CMA-ES	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E	Cv.	$F(\boldsymbol{x})$	E
Ackley	100	-	18,312	100	-	26,587	100	-	43,212
Griewangk	80	7.4E-3	17,268	100	-	25,112	100	-	38,962
Sphere	100	-	11,530	100	-	17,077	100	-	28,182

Tabla 4.5: Media de los resultados obtenidos al ejecutar 30 veces cada algoritmo con cada función objetivo. Las columnas *Cv.*, $F(\boldsymbol{x})$ y *E* representan el porcentaje de convergencia, la media de los valores fitness obtenidos y la media de evaluaciones realizadas respectivamente.

nidas de este análisis, y compararlos con los EDAs continuos y Estrategias Evolutivas (ES). Esta sección describe los experimentos realizados y los resultados obtenidos.

En este caso hemos elegido EDAs continuos que tienen en cuenta diferente número de dependencias entre las variables: UMDA$_c$, MIMIC$_c$, EGNA$_{ee}$ y CMA-ES.

Los experimentos se han realizado con individuos de 100 variables ($n = 100$) y con

tres problemas de optimización: Ackley, Griewangk y Sphere. Con el fin de realizar una comparación ecuánime entre los diferentes EBCOAs, EDAs y CMA-ES, hemos utilizado diferentes tamaños de población. La tabla 4.5 presenta los resultados obtenidos con los tamaños de población R que mejores resultados alcanzaban en cada uno de los algoritmos.

En el caso de EBCOAs, establecemos los mismos parámetros que en los experimentos anteriores: $|K| = 3$ y $|C| = 2$, el tamaño escogido para cada una de las clases fue de $R/3$ (R es el tamaño la población). En el caso de los EDAs continuos, el aprendizaje del modelo gráfico probabilístico se hace seleccionando los $R/2$ mejores individuos de la población y aplicando un enfoque elitista. Por último, para CMA-ES hemos establecido $\mu = \lambda/2$ (número máximo de padres), siendo λ el tamaño de la población (utilizando la notación original del autor), mientras que para el resto de los parámetros se mantienen los valores por defecto sugeridos originalmente por los autores.

El criterio de parada para todos los algoritmos consiste en alcanzar el óptimo (con un error menor al 10^{-6}), o haber realizado un máximo de 60000 evaluaciones.

Cada algoritmo se ha ejecutado 30 veces con cada combinación de parámetros y con cada uno de los problemas de optimización. Los resultados que se muestran en la Tabla 4.5 son los valores medios de los obtenidos en las 30 ejecuciones para cada uno de los problemas y algoritmos. Con el fin de ilustrar con mayor claridad el rendimiento de los diferentes algoritmos, estos valores medios se han calculado de la siguiente manera:

- $Cv.$ indica, en porcentaje, la tasa de convergencia.
- E o número de evaluaciones, se calculó teniendo en cuenta sólo aquellas ejecuciones en las que el algoritmo converge, es decir, encuentra el óptimo.
- $F(\boldsymbol{x})$ o valor obtenido en la función objetivo, sólo incluye las ejecuciones que alcanzaron el número máximo de evaluaciones sin converger.

Estos resultados muestran que en general, los EBCOAs resuelven bastante bien los diferentes problemas de optimización, en particular el $EBCOA_{TAN}$. También se observa que el comportamiento de los EDAs es bastante pobre, al no ser capaz de converger en la mayoría de los problemas. Sólo el más simple, UMDA$_c$, obtiene resultados aceptables con una población de 200 individuos. Estos resultados no son sorprendentes teniendo en cuenta las características de los EDAs, ya que normalmente necesitan gran número de generaciones para converger y en este caso, el número de evaluaciones se ha limitado a 60000.

En cuanto a CMA-ES, debemos decir que es la opción que mejor rendimiento muestra de todos los algoritmos, alcanzando una tasa de convergencia cercana al 100 en casi todos los problemas. Los resultados consiguen ser competitivos debido a que $EBCOA_{TAN}$ obtuvo el óptimo para todos los problemas y en la mayoría de los tamaños de población, aunque el número de evaluaciones necesarias para la convergencia es ligeramente mayor que en CMA-ES.

Por último, se ha realizado un cálculo del coste de los diferentes algoritmos. Los experimentos con EDAs y EBCOAs se han realizado en una máquina UNIX con un procesador de 2,7 GHz, 16 GB de RAM y 512 KB de Cache. Estos algoritmos fueron implementados en C++. CMA-ES se implementa en Matlab, y para ejecutarlo hemos utilizado un PC bajo Windows XP con un procesador de 3 GHz, 1GB de RAM y 512 KB de Cache. Los mejores tiempos de cómputo se obtuvieron con $EBCOA_{NB}$, donde cada ejecución duró aproximadamente 5 segundos en todos los casos. EDA_{UMDA} y EDA_{MIMIC} utilizaron aproximadamente 15 segundos, y CMA-ES necesitó cerca de 20 segundos. En el caso de $EBCOA_{TAN}$, los tiempos de ejecución son mayores, alrededor de 40 segundos cada uno. EDA_{EGNAee} es el algoritmo más lento, ya que cada ejecución puede durar más de una hora, en nuestros experimentos estas ejecuciones necesitaron entre 27 minutos y un máximo de 1h y 58 minutos.

4.5 Conclusiones

En este capítulo se introducen los fundamentos teóricos, así como el pseudocódigo genérico de un nuevo método de optimización mediante computación evolutiva basado en clasificadores Bayesianos (EBCOAs). Este método combina las técnicas de computación evolutiva con clasificadores Bayesianos a fin de resolver problemas de optimización. También se han generalizado los EBCOAs al dominio continuo mediante la aplicación de clasificadores Bayesianos adaptados para este tipo de dominios.

El comportamiento de estos algoritmos se ilustra con experimentos realizados en problemas de optimización estándar en dominios discretos como HIFF, IsoPeak y IsoTorus, así como otros problemas de optimización que utilizan variables continuas. Los resultados experimentales para comprobar el comportamiento de este nuevo enfoque se han comparado con los resultados obtenidos por otras técnicas de computación evolutiva como algoritmos genéticos, estrategias evolutivas y EDAs.

Los resultados obtenidos en los experimentos realizados en el entorno discreto muestran que el clasificador *tree augmented naïve Bayes* obtiene muy buenos resultados en todos los problemas resueltos, incluso mejora los resultados obtenidos en muchos EDAs y GAs. Además, si comparamos el comportamiento de estos algoritmos utilizando el clasificador Naïve Bayes con el algoritmo $UMDA$, su equivalente en EDAs teniendo en cuenta las dependencias entre las variables, vemos que los resultados son al menos comparables.

En cuanto a los resultados obtenidos con funciones continuas, podemos concluir que los EBCOAs continuos han demostrado ser un nuevo método capaz de obtener resultados comparables como EDAs continuos y ES. Por otra parte, los EBCOAs parecen ser muy sensibles a los parámetros de funcionamiento, lo que requiere algunas condiciones concretas para asegurar un buen rendimiento, como por ejemplo, un tamaño de la población adecuado. Para comprobarlo, se ha realizado un análisis sobre la influencia de determinados parámetros tales como el tamaño de población y la forma

en que se realiza la evolución de una población a la siguiente, en el rendimiento de estos nuevos algoritmos. Todos estos aspectos han sido analizados y probados, demostrando que un buen ajuste de los diferentes parámetros supone una mejora considerable en el rendimiento de este método.

En resumen, los resultados obtenidos a través de estos experimentos han sido útiles para demostrar que los EBCOAs pueden resultar un nuevo enfoque prometedor para mejorar el rendimiento de los EDAs, y que con un ajuste adecuado de sus parámetros pueden mostrar un rendimiento mejor que los EDAs.

Capítulo 5

Afrontando la resolución del problema de la Satisfactibilidad mediante EDAs y EBCOAs

El problema de la Satisfactibilidad (SAT) es un problema NP-completo (Cook, 1971; Garey y Johnson, 1979) que proporciona un modelo genérico mediante el cual se pueden representar otros muchos problemas reales y complejos en diferentes campos, sobre todo problemas de decisión, tales como verificación de circuitos integrados (Kuehlmann y col., 2001; Ganai y col., 2002), diseño de circuitos asíncronos (Gu y Puri, 1995) y planificación de tareas (Kautz y Selman, 1992; Giunchiglia y col., 2000). Esto ha motivado la investigación en algoritmos para la resolución del problema de la Satisfactibilidad (*SAT solvers*) durante muchos años. Existen incluso concursos a nivel internacional[1] cuyo objetivo es plantear un reto para promover nuevos algoritmos que resuelvan de forma más eficiente este problema. Se trata de algoritmos específicos diseñados para resolver SAT. También podemos encontrar en la literatura diferentes propuestas para resolver este problema aplicando algoritmos evolutivos (Gottlieb y col., 2002), principalmente GAs, aunque también EDAs (Herves y col., 2004). Sin embargo, la computación evolutiva está orientada a resolver problemas de propósito general, y por tanto no siempre puede pretender llegar a ser tan eficiente como un algoritmo específico, como ocurre en el caso del SAT.

En este capítulo nos proponemos estudiar la forma de afrontar la resolución de este problema utilizando EDAs y EBCOAs y comprobar la eficiencia de estos algoritmos para este problema concreto, con la intención de mejorar su rendimiento lo máximo posible. El objetivo de este trabajo no es competir con los algoritmos no-evolutivos, sino comprobar cómo la computación evolutiva en general, y los EDAs en particular, puede adaptarse a cualquier tipo de problema complejo. El problema SAT es ampliamente reconocido como genérico y complejo.

[1] *The International SAT Competitions. http://www.satcompetition.org/*

El capítulo comienza con una definición formal del problema SAT, así como una revisión general sobre los algoritmos que se han desarrollado para su resolución. En la sección 5.3 se analiza cómo adaptar la computación evolutiva al problema SAT. Las primeras opciones que se presentan, se basan en representaciones que ya han sido utilizadas anteriormente en la resolución del SAT mediante GAs y que en nuestro caso adaptaremos a EDAs y EBCOAs (sección 5.4). En esta sección también se incluye la utilización de técnicas adicionales como optimización local y adaptación de la función objetivo de forma dinámica. A continuación, en la sección 5.5, planteamos una nueva forma de representar el problema basada en permutaciones que nos permite realizar un pre-proceso teniendo en cuenta características propias del problema concreto a resolver. Por último se presentan diferentes opciones de hibridación de EDAs (sección 5.6) con algoritmos específicos de optimización local, de manera que ambas técnicas puedan colaborar en la resolución de problemas SAT más complejos. Para comprobar el rendimiento de las diferentes opciones, se han realizado toda una serie de experimentos que se comentan en la sección 5.7 junto a los resultados obtenidos. La sección 5.8 recoge las conclusiones generales sobre el trabajo realizado en este capítulo.

5.1 El problema booleano de la satisfactibilidad (SAT)

Formalmente, el problema de la Satisfactibilidad (SAT) está basado en un conjunto de variables Booleanas $\mathbf{Z} = (Z_1, \ldots, Z_v)$, y una fórmula Booleana $g: B^v \rightarrow B, B = \{0, 1\}$. La fórmula g se dice que es satisfactible si existe una asignación $\mathbf{z}$ que haga cierta g, e insatisfactible en caso contrario.

La mayor parte de los algoritmos diseñados para resolver SAT (*SAT solvers*) funcionan sobre problemas para los cuales g se expresa en forma normal conjuntiva (CNF). La formula g está en CNF si es una conjunción de cláusulas, es decir $g(\mathbf{z}) = c_1(\mathbf{z}) \wedge \ldots \wedge c_u(\mathbf{z})$, donde u es el número de cláusulas del problema y cada cláusula $c_i(\mathbf{z})$, con $i = 1, \ldots, u$, es una disyunción de literales. El literal es la unidad lógica fundamental del problema, que consiste simplemente en una variable o su negación. Todas las funciones Booleanas se pueden expresar en formato CNF. La ventaja de CNF consiste en que de esta forma, para que una fórmula g sea satisfecha, es suficiente con que lo sea cada cláusula individual. Para satisfacer una cláusula, es necesario que la asignación propuesta satisfaga al menos uno de los literales de la cláusula.

La variable Z_j, con $j = 1, \ldots, v$, da lugar a dos literales: el literal z_j, que se satisface al asignar a la variable Z_j el valor 1, y al literal $\bar{z}_j$, que se satisface al tomar la variable Z_j el valor 0. El objetivo es analizar y comprobar la existencia de una asignación $\mathbf{z} = (z_1, \ldots, z_v) \in B^v$ tal que $g(\mathbf{z}) = 1$. En este caso se dice que el problema SAT es *satisfactible* e *insatisfactible* en caso contrario.

La figura 5.1 muestra un ejemplo de problema SAT con cuatro variables y tres cláusulas. La función $g(z)$ será satisfactible si se satisfacen todas sus cláusulas y cada cláusula se satisface si lo hace al menos uno de sus literales. La asignación de variables

Supongamos la siguiente definición de un problema SAT:

$g(\mathbf{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \wedge (z_2 \vee \bar{z}_3 \vee z_4) \wedge (\bar{z}_1 \vee z_2 \vee \bar{z}_3)$,

$$c_1(\mathbf{z}) = (z_1 \vee z_3 \vee \bar{z}_4)$$
$$c_2(\mathbf{z}) = (z_2 \vee \bar{z}_3 \vee z_4)$$
$$c_3(\mathbf{z}) = (\bar{z}_1 \vee z_2 \vee \bar{z}_3)$$

La instancia, $\mathbf{z} = (1, 1, 0, 1) \Rightarrow z_1 = 1, z_2 = 1, z_3 = 0$ y $z_4 = 1$, satisface la función, ya que z_1 satisface $c_1(\mathbf{z})$ y tanto z_2 como $\bar{z}_3$ satifacen $c_2(\mathbf{z})$ y $c_3(\mathbf{z})$

Figura 5.1: Ejemplo de problema SAT con cuatro variables y tres cláusulas

$z_1 = 1, z_2 = 1, z_3 = 0$ y $z_4 = 1$ satisface la función ya que satisface todas las cláusulas.

Se dice que un problema SAT, es de clase k-SAT si contiene en cada cláusula exactamente k literales distintos. Mientras que la clase *2*-SAT es resoluble en un tiempo de orden polinomial, para $k \geq 3$ este problema es NP-completo (Cook, 1971).

5.2 Estado del arte en algoritmos para resolver SAT (*SAT solver*)

Los algoritmos utilizados históricamente en la resolución de SAT pueden clasificarse en dos grandes grupos: algoritmos completos y algoritmos incompletos. Los algoritmos completos exploran todo el espacio de búsqueda y la mayoría de ellos se basan en el método de Davis-Putnam-Logemann-Loveland (DPLL) (Davis y Putnam, 1960; Davis y col., 1962). El problema de estos algoritmos es que, al explorar todo el espacio de búsqueda, tienen una complejidad exponencial y por lo tanto, el tiempo necesario para la resolución de problemas muy grandes puede llegar a ser excesivamente alto. Los algoritmos incompletos no analizan la totalidad del espacio de búsqueda, explorando regiones concretas de este espacio de búsqueda. La mayoría de los algoritmos incompletos planteados en la literatura, utilizan búsqueda local o algoritmos evolutivos (EA).

Los métodos completos puede encontrar una solución satisfactoria o demostrar que no existe ninguna solución satisfactoria. Los métodos incompletos, al no explorar todo el espacio de búsqueda, no pueden probar que una fórmula es insatisfactible. La ventaja de los algoritmos incompletos es que son capaces de resolver formulas satisfactibles, que por su dimensión, no pueden resolverse con métodos completos.

En cualquier caso, los métodos de resolución de SAT más eficientes tienen en cuenta características propias del problema a resolver. A continuación se presenta un resu-

men de los métodos completos e incompletos más destacados, con el fin de analizar las características del problema que más influyen en su resolución y posteriormente trasladar estas estrategias a los EDAs.

Métodos completos

La mayoría de los algoritmos completos desarrollados para resolver el problema SAT se basan en el procedimiento presentado por Davis, Putnam, Logemann y Loveland (Davis y col., 1962), conocido como DPLL el cual es una mejora del presentado anteriormente por Davis y Putnam (Davis y Putnam, 1960), conocido como DP. Como ejemplo de este tipo de algoritmos tenemos POSIT (Freeman, 1995), SatZ (Li y Anbulagan, 1997), SATO (Zhang, 1997), GRASP (Marques-Silva y Sakallah, 1999), Chaff (Moskewicz y col., 2001) y BerkMin (Goldberg y Novikov, 2002) por citar los más conocidos.

El algoritmo Davis-Putnam-Logemann-Loveland (DPLL) comienza buscando aquellas cláusulas que contienen un solo literal y asigna, a ese literal, el valor que satisface la cláusula correspondiente. De esta manera, podemos eliminar todas las cláusulas originales que contengan el literal asignado y en aquellas cláusulas que contengan el literal complementario, eliminarlo, reduciendo el tamaño de la cláusula. Esto último puede dar lugar a nuevas cláusulas con un literal único a las que podemos aplicar esta misma regla de manera reiterada, pero también puede dar lugar a cláusulas vacías. Una cláusula vacía es una cláusula de la que se han eliminado todos los literales y su valor es cero. En este caso, nos encontramos con un *conflicto*.

En los casos en los que no existen cláusulas unitarias se aplica *la regla de la ramificación*. Esta regla elige una variable Z_j, que exista en la formula original g y reduce el problema a encontrar la satisfactibilidad de $g \bigcup z_i$ y $g \bigcup \bar{z}_i$. De esta manera se reduce el problema a dos más sencillos, ya que en cada uno de los casos se eliminan todas aquellas cláusulas que contienen el literal correspondiente y se reduce el literal complementario de las cláusulas que lo contengan. Este paso puede dar lugar a nuevas cláusulas unitarias o puede crear cláusulas vacías.

En caso de *conflicto*, es decir, cuando se crean cláusulas vacías en alguno de los pasos anteriores, el algoritmo DPLL da un paso atrás y reinicia la búsqueda en la primera rama no explorada (*backtracking*). El algoritmo termina cuando el problema no tiene cláusulas (problema satisfactible) o cuando todas las ramas generan una cláusula vacía, lo cual indica que no existe una instanciación que satisfaga el problema.

Los aspectos más importantes para la eficiencia de este proceso son la selección del literal y la resolución de conflictos. La selección del literal incluye la elección de una variable y del valor que vamos a asignar a esa variable, ya que un literal es una variable o su negación. La forma en que se tratan dichos aspectos es fundamental ya que el tiempo necesario para resolver el problema puede variar sustancialmente.

En el caso de la selección del literal, se puede elegir el literal que más veces aparece en las cláusulas de menor tamaño o el literal que más cláusulas unitarias genera.

Algunos algoritmos posteriores, han desarrollado nuevas heurísticas más complejas para la selección del literal apropiado, donde se tienen en cuenta conceptos conocidos como la *actividad de la variable* y la *conectividad de la cláusula* (Durairaj y Kalla, 2004). Este es el caso de SatZ (Li y Anbulagan, 1997) y Chaff (Moskewicz y col., 2001), dos algoritmos completos que han obtenido muy buenos resultados en la resolución de SAT. El análisis de la *actividad de la variable* y la *conectividad de la cláusula* proporcionan información útil en la solución del problema, ya que son consideradas medidas cualitativas y cuantitativas de la dependencia variable-cláusula. La actividad de la variable se define como el número de veces que aparece esa variable en todas las cláusulas, independientemente del signo con el que aparezcan. Dos cláusulas se dice que están *conectadas* si tienen una o más variables comunes (Durairaj y Kalla, 2006).

Con respecto a la resolución de conflictos, una opción consiste en eliminar la última decisión (*backtracking cronológico*). Sin embargo, la mayoría de los *SAT solver* competitivos realizan algún tipo de análisis del conflicto para obtener información que les permita realizar un backtracking no-cronológico a niveles anteriores del árbol, podando importantes regiones del espacio de búsqueda y por consiguiente, identificando las asignaciones necesarias para encontrar la solución en un tiempo menor. Como ejemplo de algoritmos que realizan análisis de conflictos podemos mencionar GRASP (Marques-Silva y Sakallah, 1996, 1999) y Chaff (Moskewicz y col., 2001).

En este sentido, también es importante destacar las aportaciones de métodos como SATO (Zhang, 1997), que incorpora estructuras de datos perezosas (*lazy*). Estas estructuras se caracterizan por mantener, para cada variable, una lista de un número reducido de cláusulas en las que aparece dicha variable. Esto ayuda a identificar si una cláusula se satisface o no o si es unitaria.

Otros métodos completos como POSIT (Freeman, 1995), SatZ (Li y Anbulagan, 1997) y GRASP (Marques-Silva y Sakallah, 1999) utilizan estructuras de datos basadas en contadores. En este caso, las cláusulas se representan como listas de literales y para cada variable, se mantiene una lista de todas las cláusulas en las que aparece dicha variable.

Entre los SAT solvers desarrollados actualmente se pueden destacar Minisat (Sörensson y Een, 2005) y RSat (Pipatsrisawat y Darwiche, 2007). Minisat es un algoritmo impulsado por la resolución del conflicto y tuvo un importante éxito en la competición sobre SAT del 2005. Su código se encuentra disponible como software libre y sólo tiene alrededor de 600 líneas de código. RSat es un algoritmo completo para resolver SAT basado en el método DPLL que, al igual que Minisat, utiliza modernos heurísticos basados en las ideas de Chaff para la selección del literal, en los que se tiene en cuenta la actividad de la variable. RSat ganó la medalla de oro de la competición de SAT de 2007[2] en la categoría industrial. La versión anterior de RSat obtuvo el tercer lugar en la competición de SAT de 2006.

[2] *http://reasoning.cs.ucla.edu/rsat/*

Métodos incompletos

La mayoría de algoritmos incompletos existentes para SAT se basan generalmente en la búsqueda local estocástica, a veces denominada SLS (*stochastic local search*) (Kautz y col., 2009).

Los algoritmos de búsqueda local comienzan generando una interpretación de forma aleatoria e intentan encontrar una solución que satisfaga la formula cambiando, en cada iteración, el valor de alguna de las variables. Estos cambios se repiten hasta encontrar una solución satisfactible o hasta realizar un número de cambios determinado. Si después de un determinado número de intentos no se ha encontrado una solución, se repite el proceso volviendo a reiniciar el algoritmo con una nueva asignación, para salir de mínimos locales. Esta repetición del proceso se realiza un número determinado de veces. Los diferentes algoritmos de búsqueda local se diferencian en la forma de elegir la variable a la que se cambia su valor. Dos ejemplos de algoritmos incompletos de búsqueda local muy referenciados en la literatura son GSAT (Selman y col., 1992) y su potente variación Walksat (Selman y col., 1994).

GSAT comienza generando una interpretación de manera aleatoria e intenta mejorarla paso a paso. En cada iteración cambia el valor de una variable en la interpretación actual. A la hora de elegir qué variable se va a cambiar, lo hace entre aquellas que aumenten el número de cláusulas satisfechas. Cuando se han realizado un determinado número de cambios, se reinicia la búsqueda intentando huir de óptimos locales. Este proceso se realiza un máximo número de intentos.

Walksat sigue el mismo esquema pero utiliza otro criterio a la hora de seleccionar la variable a la que cambiará su valor en cada iteración: selecciona una cláusula de entre las no satisfechas y cambia una de las variables de esa cláusula. En este caso hay que comprobar que al modificar esta variable no se rompa alguna de las cláusulas ya satisfechas anteriormente. WalkSat normalmente es más eficiente que GSAT ya que consigue evitar los mínimos locales con mayor eficiencia (Selman y col., 1996). En algunos artículos se refiere a WalkSat, como WSAT (de Walk-SAT) (Selman y col., 1996).

Otros ejemplos de algoritmos que siguen este mismo esquema son Novelty y R-Novelty (McAllester y col., 1997) aunque utilizan diferentes criterios a la hora de seleccionar las variables.

Los métodos basados en algoritmos incompletos son más rápidos que los completos ya que no exploran, en general, todo el espacio de búsqueda. Esto les permite resolver problemas, que por su tamaño resultan prohibitivos para un algoritmo completo. Sin embargo, un algoritmo incompleto no puede demostrar que un problema no es satisfactible. Estas razones han provocado la hibridación de algoritmos donde se combina búsqueda local con algoritmos completos basados en el enfoque DPLL (Ferris y Froehlich, 2004; Fang y Hsiao, 2007; Letombe y Marques-Silva, 2008; Wang y col., 2008).

Los primeros intentos de resolver SAT mediante algoritmos evolutivos, principal-

mente mediante GAs, confirman la dificultad de estos algoritmos para competir con algoritmos específicos para SAT (de Jong y Spears, 1989). Además, Rana y Whitley en 1998 demostraron que los GAs clásicos resultan inadecuados para la función de idoneidad MAXSAT (que cuenta el número de cláusulas satisfechas), ya que esta función obtiene valores engañosos al asignar el mismo valor a puntos muy diferentes del espacio del búsqueda (Rana y Whitley, 1998). Todo ello ha llevado a aplicar técnicas adicionales al proceso evolutivo con el fin de conseguir resultados más eficientes. Estas técnicas adicionales incluyen adaptar la función objetivo, utilizar operadores de mutación específicos para el problema así como añadir procedimientos de optimización local. A la hora de adaptar la función objetivo, se tiene en cuenta la dificultad de satisfacer determinadas cláusulas, incluyendo esta dificultad como un peso adicional en la función objetivo. SAWEA (Eiben y van der Hauw, 1997), RFEA (Gottlieb y Voss, 1998a, 2000) y sus versiones RFEA2 y RFEA+ (Gottlieb y Voss, 2000) son ejemplos basados en adaptación de la función objetivo y uso específico de operadores de mutación. FlipGA (Marchiori y Rossi, 1999) y ASAP (Rossi y col., 2000) son ejemplos representativos de algoritmos genéticos para SAT que utilizan la función MAXSAT y añaden optimización local. Esta optimización local consiste en cambiar el valor de algunas variables de individuos con un buen valor fitness, para rastrear más profundamente el espacio de búsqueda. GASAT (Hao y col., 2003; Lardeux y col., 2006) es un algoritmo híbrido basado en operadores de cruce específicos para SAT, combinado con búsqueda Tabu.

Los algoritmos mencionados anteriormente están diseñados de manera específica para resolver SAT. Sin embargo, también podemos encontrar intentos de resolver SAT utilizando otro tipo de algoritmos pensados para resolver problemas de optimización en general, como es el caso de hBOA(*hierarchical bayesian optimization algorithm*) (Pelikan y Goldberg, 2003).

5.3 Caracterización de problemas SAT mediante computación evolutiva

A la hora de resolver cualquier problema mediante algoritmos evolutivos (EA) es necesario encontrar una forma de representar cada posible solución y definir una función de evaluación que aplicada a una solución concreta sea capaz de valorar cómo de buena es esa solución en relación a otras posibles. Estos dos componentes resultan críticos para que el algoritmo evolucione favorablemente. En el caso concreto de SAT, también es importante ayudar al proceso evolutivo mediante técnicas adicionales. Además de las técnicas comentadas en el apartado anterior, otra posibilidad a tener en cuenta para que el proceso evolutivo sea más eficiente, consiste en considerar determinadas características del problema concreto tales como la actividad de la variable y la conectividad de la cláusula. En este caso, se puede hacer un estudio previo de estas características (*pre-proceso*) y utilizarlo para realizar una inicialización específica de

la población inicial, en vez de realizar esta inicialización al azar.

En este apartado se presenta un estudio de todos estos aspectos y su influencia en el rendimiento de los EDAs y EBCOAs a la hora de resolver un problema SAT. En primer lugar se analizan diferentes representaciones utilizadas en la resolución de SAT mediante computación evolutiva y en cada caso se comprueba la posibilidad de incorporar técnicas de pre-proceso, adaptación de la función objetivo y optimización local.

5.4 Tipos de representaciones y su función fitness asociada

Cada una de las representaciones que se presentan a continuación sugiere una función de evaluación diferente que marque cuando una solución es mejor que otra. En las tres primeras, las variables del individuo son discretas: la primera de ellas almacena el valor que toma cada una de las variables del problema mientras que las otras se fijan en las cláusulas y en cómo se satisfacen. Además, se presentan otras opciones en las que las variables del individuo son continuas. Para cada tipo de representación, se plantea la función objetivo apropiada para guiar el proceso de búsqueda.

5.4.1 Representación mediante cadena (string) de bits

Consiste en representar la solución candidata mediante una secuencia de bits de longitud v, siendo v el número de variables del problema SAT que intentamos resolver, de manera que cada variable del individuo X_i, con $i = 1, \ldots, v$, se asocia con un bit.

Es importante considerar la función fitness apropiada. La función Booleana original $g(\boldsymbol{z})$, no dirige la búsqueda. La función de evaluación $F_{MAXSAT}(\boldsymbol{x})$ consiste en contar el número de cláusulas satisfechas, tratando de maximizar este valor. En la formulación MAXSAT, el valor objetivo es equivalente al número de cláusulas satisfechas y el óptimo se alcanza cuando este valor coincide con el número de cláusulas del problema. Es decir, cuando se satisfacen todas las cláusulas. En este caso, las variables del individuo, X_i, coinciden con las variables del problema SAT, Z_i.

$$F_{MAXSAT}(\boldsymbol{x}) = c_1(\boldsymbol{x}) + \ldots + c_u(\boldsymbol{x}) \tag{5.1}$$

donde $c_j(\boldsymbol{x})$, con $j = 1, \ldots, u$, representa el valor de la cláusula j-ésima y u es el número de cláusulas del problema.

$$c_i(\boldsymbol{x}) = \begin{cases} 0 & \text{si no se satisface la cláusula} \\ 1 & \text{si la cláusula se satisface} \end{cases} \tag{5.2}$$

La figura 5.2 muestra un ejemplo de este tipo de representación en el que se evalúan dos individuos distintos para la fórmula SAT planteada en el ejemplo de la figura 5.1.

$$g(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \wedge (z_2 \vee \bar{z}_3 \vee z_4) \wedge (\bar{z}_1 \vee z_2 \vee \bar{z}_3)$$

Individuo: $(1,0,1,0) \Rightarrow z_1 = 1, z_2 = 0, z_3 = 1, z_4 = 0$

$$\left.\begin{array}{l} c_1(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) = 1 \rightarrow \text{resuelven } z_1 = 1, z_3 = 1 \text{ y } z_4 = 0 \\ c_2(\boldsymbol{z}) = (z_2 \vee \bar{z}_3 \vee z_4) = 0 \rightarrow \text{no se resuelve con este individuo} \\ c_3(\boldsymbol{z}) = (\bar{z}_1 \vee z_2 \vee \bar{z}_3) = 0 \rightarrow \text{no se resuelve con este individuo} \end{array}\right\} \rightarrow g(\boldsymbol{z}) = 0$$

$$F_{MAXSAT}(\boldsymbol{x}) = c_1(\boldsymbol{z}) + c_2(\boldsymbol{z}) + c_3(\boldsymbol{z}) = 1 + 0 + 0 = 1$$

Individuo: $(1,0,0,0) \Rightarrow z_1 = 1, z_2 = 0, z_3 = 0, z_4 = 0$

$$\left.\begin{array}{l} c_1(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) = 1 \rightarrow \text{resuelven } z_1 = 1 \text{ y } z_4 = 0 \\ c_2(\boldsymbol{z}) = (z_2 \vee \bar{z}_3 \vee z_4) = 1 \rightarrow \text{resuelven } z_3 = 0 \\ c_3(\boldsymbol{z}) = (\bar{z}_1 \vee z_2 \vee \bar{z}_3) = 1 \rightarrow \text{resuelven } z_3 = 0 \end{array}\right\} \rightarrow g(\boldsymbol{z}) = 1$$

$$F_{MAXSAT}(\boldsymbol{x}) = c_1(\boldsymbol{z}) + c_2(\boldsymbol{z}) + c_3(\boldsymbol{z}) = 1 + 1 + 1 = 3$$

Figura 5.2: Ejemplo de problema SAT utilizando la representación mediante cadena de bits

En el primero de ellos, con el individuo (1, 0, 1, 0), $g(\boldsymbol{z}) = 0$ ya que no satisface las cláusulas segunda y tercera. La función objetivo $F_{MAXSAT}(\boldsymbol{x})$ toma el valor 1 mostrando el número de cláusulas satisfechas. En el segundo caso, con el individuo (1, 0, 0, 0) se satisfacen todas las cláusulas, por lo que $g(\boldsymbol{z}) = 1$ y la función objetivo $F_{MAXSAT}(\boldsymbol{x})$ toma el valor 3.

Esta es la opción utilizada en la mayoría de las publicaciones sobre resolución de SAT mediante Algoritmos Evolutivos. La mayor desventaja está en que la función objetivo MAXSAT, asigna el mismo valor a individuos muy diferentes y distantes dentro del espacio de búsqueda. Es decir, esta función confunde al algoritmo en la búsqueda de cómo converger hacia el óptimo.

5.4.2 Representación del camino *(path)*

Esta representación, sugerida en (Gottlieb y Voss, 1998b), se basa en la idea de que para encontrar una solución satisfactoria sólo es necesario satisfacer un literal de cada cláusula. De esta manera, para crear una posible solución (un individuo) es suficiente con seleccionar un literal de cada cláusula, generando un camino que recorre todas las cláusulas, visitando una cláusula cada vez. Un individuo se compone de tantas variables como cláusulas. En este caso es importante diferenciar las variables del individuo X_i, con $i = 1, \ldots, u$ donde u es el número de cláusulas, de las variables del problema SAT, Z_j, con $j = 1, \ldots, v$ donde v el número de variables del problema SAT que intentamos resolver. Cada variable del individuo X_i, puede tomar un valor de $1, \ldots, k$ (k-SAT) que indica el literal que se ha elegido para satisfacer la cláusula correspondiente. Es decir, el valor de la variable del individuo indica una variable del problema y el valor

$$g(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \wedge (z_2 \vee \bar{z}_3 \vee z_4) \wedge (\bar{z}_1 \vee z_2 \vee \bar{z}_3)$$

Individuo: $\boldsymbol{x} = (1, 1, 3) \Rightarrow x_1 = 1, x_2 = 1, x_3 = 3$

$x_1 = 1, \quad c_1(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \quad \Rightarrow \quad z_1 = 1$

$x_2 = 1, \quad c_2(\boldsymbol{z}) = (z_2 \vee \bar{z}_3 \vee z_4) \quad \Rightarrow \quad z_2 = 1$

$x_3 = 3, \quad c_3(\boldsymbol{z}) = (\bar{z}_1 \vee z_2 \vee \bar{z}_3) \quad \Rightarrow \quad z_3 = 0$

No existen inconsistencias $\Rightarrow F(\boldsymbol{x}) = 0$

Individuo: $\boldsymbol{x} = (1, 3, 1) \Rightarrow x_1 = 1, x_2 = 3, x_3 = 1$

$x_1 = 1, \quad c_1(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \quad \Rightarrow \quad z_1 = 1$

$x_2 = 1, \quad c_2(\boldsymbol{z}) = (z_2 \vee \bar{z}_3 \vee z_4) \quad \Rightarrow \quad z_4 = 1$

$x_3 = 3, \quad c_3(\boldsymbol{z}) = (\bar{z}_1 \vee z_2 \vee \bar{z}_3) \quad \Rightarrow \quad z_1 = 0$

Existe una inconsistencia con $z_1 \Rightarrow F(\boldsymbol{x}) = 1$

Figura 5.3: Ejemplo de problema SAT utilizando la representación del camino (*path*)

que se le asignará a dicha variable. Si $X_i = 2$ y el segundo literal de la cláusula i-ésima es $\bar{z_j}$, eso quiere decir que a la variable Z_j se le asignará el valor 0.

Sin embargo, en esta representación pueden aparecer inconsistencias. Una inconsistencia se da cuando se ha elegido la misma variable para satisfacer dos cláusulas distintas y en una cláusula se encuentra negada y en la otra no. Las inconsistencias equivalen a los *conflictos* mencionados al describir los métodos completos en la sección 5.2. Una función objetivo razonable para esta representación debe medir la cantidad de inconsistencias y se debe de minimizar este valor.

Si el camino con menor número de inconsistencias tiene al menos una inconsistencia, la solución no satisface el problema SAT. Si existe un camino sin inconsistencias existe al menos una asignación de variables que satisface el problema SAT.

La figura 5.3 muestra un ejemplo de este tipo de representación con dos individuos distintos para la formula SAT planteada en el ejemplo de la figura 5.1. El primero de ellos, (1, 1, 3), indica que la primera cláusula se resuelve con el primer literal de esa cláusula ($z_1 = 1$), la segunda cláusula, también con el primer literal ($z_2 = 1$) y la tercera cláusula con el tercer literal ($z_3 = 0$), mientras que z_4 puede tomar indistintamente el valor 1 ó 0. Estos valores de las variables satisfacen el problema SAT y no presentan inconsistencias, por lo que la función objetivo tomará el valor 0 ($F(\boldsymbol{x}) = 0$). En el segundo caso, el individuo (1, 3, 1), indica que la primera cláusula se resuelve con el primer literal de esa cláusula ($z_1 = 1$), la segunda cláusula con el tercer literal ($z_4 = 1$) y la tercera cláusula con el primer literal ($z_1 = 0$). Esta asignación es imposible ya que contiene una inconsistencia para z_1 tal y como se refleja en el valor de la función objetivo $F(\boldsymbol{x}) = 1$.

En esta opción se tiene en cuenta la importancia de la cláusula, de manera que sea esta idea la que guíe el proceso de búsqueda. Sin embargo, la relación variable-cláusula, es decir, en cuantas cláusulas aparece cada variable y con que valor, es una información que puede ser relevante y que no se tiene en cuenta en este caso.

5.4.3 Representación de cláusula

En (Hao, 1995) se propone la representación conocida como de cláusula, que enfatiza el efecto local de las variables en las cláusulas.

En un problema 3-SAT, cada cláusula contiene 3 literales y cada literal representa un posible valor de una variable booleana. Esto significa que existen $2^3 = 8$ asignaciones posibles $(000, 001, 010, 011, 100, 101, 110, 111)$ para cada cláusula, de las cuales 7 satisfacen la cláusula y sólo una de ellas no la satisface. Por ejemplo, la única asignación que no satisface la cláusula $(z_2 \vee \bar{z}_3 \vee z_4)$ es el valor 2 (010 en binario), donde $z_2 = 0, z_3 = 1$ y $z_4 = 0$. La idea consiste en seleccionar cualquiera de las 7 asignaciones factibles para cada cláusula, y apuesta por encontrar una asignación parcial de variables consistente globalmente.

En esta representación, cada individuo está formado por u variables, siendo u el número de cláusulas del problema, y cada variable del individuo toma el valor correspondiente a la representación de variables del problema seleccionada para resolver esa cláusula. Para un problema k-SAT cada variable puede tomar un valor entre 0 y $2^K - 1$. En el ejemplo anterior, si la variable del individuo correspondiente a esa cláusula toma el valor 4 (100 en binario), eso supone que $z_2 = 1, z_3 = 0$ y $z_4 = 0$.

Por lo tanto, en esta representación también es importante diferenciar entre las variables del individuo X_i, con $i = 1, \ldots, u$ donde u es el número de cláusulas, de las variables del problema SAT, Z_j, con $j = 1, \ldots, v$ donde v el número de variables del problema SAT que intentamos resolver.

Aquí también se dan inconsistencias (o *conflictos*) si se elige la misma variable con valores diferentes para resolver diferentes cláusulas. Por lo tanto, la función objetivo que se propone para esta representación debe tener en cuenta la cantidad de inconsistencias, de manera que se intente minimizar este valor.

Dado el ejemplo de la figura 5.1, las asignaciones prohibidas serían: 1 para la cláusula $c_1(\mathbf{z})$, 2 para la cláusula $c_2(\mathbf{z})$ y 5 para la cláusula $c_3(\mathbf{z})$. Lo cual corresponden al individuo (1 2 5), que es el único prohibido. La figura 5.4 muestra como se instanciarían dos individuos factibles.

Igual que en la representación anterior, si la solución con menor número de inconsistencias tiene al menos una inconsistencia, la solución no satisface el problema SAT. Si existe una solución sin inconsistencias existe al menos una asignación de variables que satisface el problema SAT.

$$g(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \wedge (z_2 \vee \bar{z}_3 \vee z_4) \wedge (\bar{z}_1 \vee z_2 \vee \bar{z}_3)$$

Individuo: $\boldsymbol{x} = (0, 1, 1)$

$x_1 = 0$ (000), $c_1(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \Rightarrow z_1 = 0, z_3 = 0, z_4 = 0$
$x_2 = 1$ (001), $c_2(\boldsymbol{z}) = (z_2 \vee \bar{z}_3 \vee z_4) \Rightarrow z_2 = 0, z_3 = 0, z_4 = 1$
$x_3 = 1$ (001), $c_3(\boldsymbol{z}) = (\bar{z}_1 \vee z_2 \vee \bar{z}_3) \Rightarrow z_1 = 0, z_2 = 0, z_3 = 1$

Existen inconsistencias con z_3 y z_4, $\Rightarrow F(\boldsymbol{x}) = 2$

Individuo: $\boldsymbol{x} = (2, 6, 3)$

$x_1 = 2$ (010), $c_1(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \Rightarrow z_1 = 0, z_3 = 1, z_4 = 0$
$x_2 = 6$ (110), $c_2(\boldsymbol{z}) = (z_2 \vee \bar{z}_3 \vee z_4) \Rightarrow z_2 = 1, z_3 = 1, z_4 = 0$
$x_3 = 3$ (011), $c_3(\boldsymbol{z}) = (\bar{z}_1 \vee z_2 \vee \bar{z}_3) \Rightarrow z_1 = 0, z_2 = 1, z_3 = 1$

No existen inconsistencias $\Rightarrow F(\boldsymbol{x}) = 0$. Solución: (0, 1, 1, 0)

Figura 5.4: Ejemplo de problema SAT utilizando la representación de la cláusula

5.4.4 Representación en coma flotante

Todas las representaciones anteriores son propuestas de optimización que utilizan valores que pertenecen a dominios discretos. Bäck y col. (Bäck y col., 1998) proponen transformar un problema SAT en un problema de optimización continuo al que se puedan aplicar estrategias evolutivas (ES). En su propuesta, las soluciones candidatas se representan mediante un vector de valores continuos $\mathbf{y} \in [-1, 1]^n$, donde se truncan los valores negativos a -1 y los positivos a 1. La función objetivo que utiliza se obtiene de transformar la fórmula SAT. Esta transformación se basa en sustituir los literales z_j y $\bar{z}_j$ por $(y_i - 1)^2$ e $(y_i + 1)^2$ respectivamente y reemplazar los operadores booleanos $\vee$ y $\wedge$ por la operaciones aritméticas $\cdot$ y $+$ respectivamente, convirtiendo la función $g : \mathbf{B}^n \rightarrow \mathbf{B}$ en $h : [-1, 1]^n \rightarrow \mathbf{R}^+$ (Bäck, 1996).

La figura 5.5 muestra un ejemplo de este tipo de representación con dos individuos reales distintos, para la fórmula SAT planteada en el ejemplo de la figura 5.1. En cada uno de ellos se han sustituido los valores negativos por -1 y los positivos por 1. Además, la ecuación original $g : \mathbf{B}^4 \rightarrow \mathbf{B}$ representada en la figura 5.1, se transforma en la siguiente función continua $h : [-1, 1]^4 \rightarrow \mathbf{R}^+$, que se utilizará como función de evaluación.

$$h(\boldsymbol{y}) = (y_1 - 1)^2(y_3 - 1)^2(y_4 + 1)^2 + (y_2 - 1)^2(y_3 + 1)^2(y_4 - 1)^2 + (y_1 + 1)^2(y_3 + 1)^2(y_2 - 1)^2$$

$$g(\boldsymbol{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \wedge (z_2 \vee \bar{z}_3 \vee z_4) \wedge (\bar{z}_1 \vee z_2 \vee \bar{z}_3)$$
se transforma en:
$$h(\boldsymbol{y}) = (y_1 - 1)^2(y_3 - 1)^2(y_4 + 1)^2 + (y_2 - 1)^2(y_3 + 1)^2(y_4 - 1)^2 + (y_1 + 1)^2(y_3 + 1)^2(y_2 - 1)^2$$

Individuo: $\boldsymbol{x} = (-0{,}4723, 0{,}6954, 0{,}3912, -0{,}8931) \Rightarrow \boldsymbol{y} = (-1, 1, 1, -1)$

$$\left.\begin{array}{l} (y_1 - 1)^2(y_3 - 1)^2(y_4 + 1)^2 = (-2)^2(0)^2(0)^2 = 0 \\ (y_2 - 1)^2(y_3 + 1)^2(y_4 - 1)^2 = (-2)^2(0)^2(0)^2 = 0 \\ (y_1 + 1)^2(y_3 + 1)^2(y_2 - 1)^2 = (-2)^2(0)^2(0)^2 = 0 \end{array}\right\} \rightarrow h(\boldsymbol{y}) = 0$$

Solución que satisface el problema SAT $\Rightarrow \boldsymbol{z} = (0, 1, 1, 0) \Rightarrow g(\boldsymbol{z}) = 0$

Individuo: $\boldsymbol{x} = (0{,}5819, -0{,}3991, 0{,}1101, -0{,}4518) \Rightarrow \boldsymbol{y} = (1, -1, 1, -1)$

$$\left.\begin{array}{l} (y_1 - 1)^2(y_3 - 1)^2(y_4 + 1)^2 = (0)^2(0)^2(0)^2 = 0 \\ (y_2 - 1)^2(y_3 + 1)^2(y_4 - 1)^2 = (-2)^2(2)^2(-2)^2 \neq 0 \\ (y_1 + 1)^2(y_3 + 1)^2(y_2 - 1)^2 = (2)^2(-2)^2(2)^2 \neq 0 \end{array}\right\} \rightarrow h(\boldsymbol{y}) \neq 0$$

Solución que no satisface el problema SAT $\Rightarrow \boldsymbol{z} = (1, 0, 1, 0) \Rightarrow g(\boldsymbol{z}) = 1$

Figura 5.5: Ejemplo de problema SAT utilizando la representación en coma flotante

5.4.5 Utilización de técnicas adicionales para dirigir la búsqueda

Optimización local

Como ya se ha comentado anteriormente, la mayoría de los intentos de resolver SAT mediante computación evolutiva dan mejores resultados al completarlos con técnicas adicionales. Una opción es utilizar optimización local. Las técnicas de optimización local consisten en aplicar cambios concretos en alguna de las variables del individuo, que ayuden a dirigir el proceso hacia el óptimo. Estas técnicas son generales y pueden utilizarse independientemente del algoritmo de búsqueda elegido así como de la representación escogida.

Inicialmente, la función objetivo guía la búsqueda de manera eficiente, pero a partir de un momento, la información que aporta esta función no es suficiente. Esto es debido a la ambigüedad que se produce debido a que individuos muy diferentes obtienen el mismo valor al ser evaluados, lo cual quiere decir que la función objetivo no proporciona una orientación suficientemente precisa para guiar al algoritmo en el espacio de búsqueda. La idea es añadir este tipo de optimización con el fin de profundizar en regiones concretas de dicho espacio. En este sentido, el algoritmo evolutivo se aplica como medio de acercarnos a regiones del espacio de búsqueda que puedan contener un óptimo, mientras que la optimización local se utiliza para concentrarse en dichas

regiones.

Si la optimización local no se incluye desde el principio, es importante decidir qué momento es el adecuado para empezar a aplicar este tipo de optimización. Una opción sensata podría ser comenzar a aplicar esta optimización cuando exista un determinado número de individuos que obtenga el mismo valor en la función objetivo, aunque también podemos optar por comenzar a partir de una determinada generación de individuos.

Un ejemplo de este tipo de optimización con algoritmos genéticos podemos encontrarlo en FlipGA (Marchiori y Rossi, 1999). La base de FlipGA consiste en aplicar el heurístico *Flip* a cada individuo después de realizar el cruce y la mutación. El heurístico explora los genes al azar y se *cambia* cada gen. Se acepta el cambio si mejora el valor de este individuo. Es decir, si aumenta el número total de cláusulas satisfechas después del cambio. Este mismo heurístico se utiliza en ASAP (Rossi y col., 2000) y en hBOA (Pelikan y Goldberg, 2003).

En estos casos, la resolución del problema SAT se debe más a la incorporación de estas técnicas de optimización local que al propio algoritmo evolutivo. De hecho, los algoritmos de búsqueda local GSAT y WalkSAT comentados anteriormente en la sección 5.2 resuelven SAT aplicando simplemente esta idea.

A continuación se presentan algunas propuestas concretas de cómo se puede aplicar este tipo de mejora en las diferentes representaciones de individuos planteadas anteriormente.

Optimización local en representación mediante secuencia de bits

Para aplicar optimización local cuando se utiliza la representación mediante secuencia de bits, descrita en la sección 5.4.1, es necesario marcar aquellas variables que pertenecen a cláusulas no satisfechas, cuando se evalúa a cada nuevo individuo generado. De esta manera, la optimización consiste en modificar el valor de cada variable marcada y volver a evaluar al individuo. Si en la nueva evaluación, después del cambio, se obtiene un mejor valor en la función objetivo, se mantendrá dicho cambio y se rechazará en caso contrario.

Optimización local en representación del camino (path)

En la representación mediante el path (sección 5.4.2), debemos recordar que las variables del individuo indican, para cada cláusula, cual es el literal, de los k literales de esa cláusula, que se utilizará para resolverla. Cuando ocurre una inconsistencia, significa que para satisfacer la cláusula actual se ha elegido un literal al que ya se le había asignado un valor y ese valor es contrario al que debería tomar para satisfacer la cláusula actual. En ese caso, en el momento en que se evalúa a un nuevo individuo, se marcan aquellas variables que provocan una inconsistencia, de manera que el proceso

de optimización modifique estas variables marcadas. Después de cada cambio se vuelve a evaluar el nuevo individuo y se decide, según el valor obtenido en la evaluación, si se mantiene el cambio o no.

Optimización local en representación de cláusula

Cuando se utiliza la representación de cláusula (sección 5.4.3), las variables del individuo también indican la forma en que se resolverá una determinada cláusula, aunque de manera diferente a la anterior. Además, la función objetivo que se utiliza en este caso también cuenta las inconsistencias que se van produciendo. Por tanto, con esta representación, se puede aplicar optimización local utilizando el mismo procedimiento que en el caso anterior.

Adaptación de la función objetivo mediante pesos

Otra posibilidad, a la hora de aplicar técnicas adicionales, consiste en adaptar la función objetivo de forma dinámica dando un mayor *peso* a aquellas cláusulas no resueltas. Si la función objetivo MAXSAT cuenta el número de cláusulas satisfechas, se puede sustituir por la ecuación: $F(\boldsymbol{x}) = w_1 \cdot c_1(\boldsymbol{x}) + \ldots + w_u \cdot c_u(\boldsymbol{x})$ donde $c_i(\boldsymbol{x})$, con $i = 1, \ldots, u$, representa el valor booleano de la cláusula i-ésima, u es el número de cláusulas del problema y w_i representa el peso de la cláusula i-ésima, que se va calculando de forma dinámica.

Inicialmente todos los pesos w_i se inicializan a 1 ($w_i = 1$), lo cual equivale a utilizar la función MAXSAT presentada en la ecuación 5.1. A lo largo de todo el proceso evolutivo, esos pesos se van actualizando de manera que se incrementan en uno los pesos correspondientes a las cláusulas no resueltas, dejando el resto con el valor actual:

$$w_i = w_i + 1 - c_i(\boldsymbol{x}^*) \tag{5.3}$$

donde $c_i(\boldsymbol{x}^*)$ indica el valor actual de la cláusula i-ésima. Esta adaptación incrementa sólo los pesos correspondientes a cláusulas no satisfechas.

Con esta adaptación se pretende ayudar implícitamente al algoritmo evolutivo a salir de óptimos locales, enfocándola a las cláusulas que resultan más difíciles de resolver, y por tanto guiando el proceso de búsqueda por los pesos.

Al aplicar esta técnica, es importante considerar en qué momento se realiza la actualización de los pesos. Puede hacerse desde el principio, empezar a aplicarla a partir de una determinada generación, o bien cuando exista un determinado número de individuos diferentes con el mismo valor fitness. Por otro lado, el vector de pesos se puede actualizar en todas las generaciones o solamente cada un número determinado de generaciones.

Podemos encontrar ejemplos en los que se ha añadido esta técnica, conocida como *SAW (stepwise adaptation of weights)*, a un algoritmo genético para resolver SAT

en (Eiben y van der Hauw, 1997). En (Bäck y col., 1998) se aplica esta técnica a algoritmos evolutivos y al algoritmo de búsqueda local GSAT.

Aunque la adaptación de pesos también puede considerarse como una técnica adicional al proceso evolutivo, a diferencia de la optimización local, que es totalmente externa, en este caso se intenta influir en el proceso de búsqueda, modificando la valoración de los individuos. En el caso concreto de los EDAs, esta adaptación obliga a volver a calcular las relaciones de dependencia entre variables, de manera que puede ayudar al proceso a salir de óptimos locales.

5.5 Representaciones utilizando permutaciones

En esta sección se plantea una nueva forma de representar el problema utilizando permutaciones, de manera que cada individuo represente un orden, concretamente el orden en que se asignarán los valores a cada una de las variables del problema SAT.

Al analizar los SAT solver completos que mejores resultados obtienen, uno de los aspectos más importantes es la elección de la variable que se resuelve en cada momento. Es decir, el orden en que se resuelven las variables siguiendo un procedimiento concreto. Además, teniendo en cuenta la dependencia variable-cláusula, también es importante el orden en que se resuelven las cláusulas. En este sentido, puede resultar muy efectivo utilizar una representación del individuo que establezca este orden y asociada a dicha representación, la función objetivo correspondiente capaz de guiar al proceso de aprendizaje hacia el orden adecuado, ayudando a encontrar una asignación que satisfaga al problema SAT original rápidamente. Una forma de establecer este orden consiste en utilizar permutaciones en la representación de los individuos.

En las representaciones comentadas anteriormente, se van asignando valores a cada una de las variables del problema y se mide, mediante la función objetivo, la bondad de estas asignaciones, evolucionando hasta encontrar la mejor. En las representaciones mediante permutaciones, las variables del individuo no representan a las variables del problema. Cada individuo representa el orden en que se asignan valores a las variables. De esta manera, el proceso encontrará aquellas cláusulas o variables que son más difíciles de resolver, de manera que se resuelvan las primeras. Resolver estas cláusulas o variables supone asignar a cada variable del problema original el valor más adecuado. Para ello, se puede utilizar información del problema relativa a la actividad de la variable, es decir, considerar cuántas cláusulas contienen a cada variable.

Permutaciones

La representación de individuos basada en permutaciones se ha aplicado normalmente a problemas tales como el del agente viajero (Flood, 1956) o el problema de rutas de vehículos (Fiala, 1978). Un ejemplo ilustrativo de la aplicación de permutaciones en algoritmos genéticos para la solución de estos problemas se puede encontrar

en (Freisleben y Merz, 1996). También podemos encontrar representaciones de individuos basadas en permutaciones para la búsqueda de correspondencia de grafos inexactos mediante EDAs en (Bengoetxea y col., 2001a). Sin embargo, no tenemos constancia de que se haya utilizado este tipo de representación en la resolución del SAT.

Una permutación es una lista de números en la que todos los valores de 1 a n tienen que aparecer en un individuo de tamaño n. Según esta definición, se está hablando de valores discretos. Sin embargo, para conseguir que los individuos de una población cumplan esta condición, sería necesario modificar el proceso de generación y simulación de los individuos de una población, tanto en los EDAs como en los EBCOAs. En vez de esto, se intentará aprovechar el proceso y convertir a cada individuo generado en una permutación. Esta conversión se puede realizar a partir de individuos discretos. Sin embargo, el proceso de convertir el individuo generado por el algoritmo en una permutación, resulta más directo si se utiliza la implementación continua de los EDAs o EBCOAs, ya que la probabilidad de que dos variables continuas de un mismo individuo tomen el mismo valor, es nula.

Por tanto, en las representaciones mediante permutaciones lo que se plantea es utilizar EDAs o EBCOAs continuos y a la hora de evaluar a cada individuo, convertirlo en una permutación que indique en que orden se resolverán las variables o cláusulas del problema. Teniendo en cuenta este orden e información propia del problema, se asignan los valores correspondientes a cada variable del problema y de esta forma se puede calcular el número de cláusulas no resueltas, que es el valor que se asignara al individuo. A continuación se describe como se puede convertir un individuo continuo en una permutación.

Conversión de un individuo continuo en una permutación

Sea u el tamaño del individuo y $\boldsymbol{x} = (x_1, \ldots, x_u)$, con $x_i \in \mathbf{R}$, el individuo del cual se quiere obtener una permutación. El proceso comienza creando el vector de enteros $\boldsymbol{p} = (p_1, \ldots, p_u)$, que contendrá la permutación, y que se inicializa con $p_i = i$, para $i = 1, \ldots, u$. A continuación se ordena el vector $\boldsymbol{x}$, de manera que cada vez que sea necesario intercambiar dos variables de este vector, se intercambian también las correspondientes variables del vector $\boldsymbol{p}$. Al final del proceso, el vector $\boldsymbol{p}$ contiene una permutación con todos los valores de 1 a u, en el orden marcado por el individuo $\boldsymbol{x}$ inicial. En la figura 5.6 se puede ver un ejemplo de este proceso en el que el orden obtenido indica que primero se debe resolver la variable X_2, a continuación la variable X_4, luego la variable X_3 y por último X_1.

Diferentes tipos de representaciones mediante premutaciones

Se pueden plantear dos opciones, según el tipo de orden que se quiera indicar con

Individuo:	0,3576	0,0139	0,2985	0,1170
Inicialización del orden:	1	2	3	4
Después de ordenar				
Individuo ordenado:	0,0139	0,1170	0,2985	0,3576
orden calculado:	2	4	3	1

Figura 5.6: Ejemplo de cómo convertir un individuo con variables continuas en el orden en que se van a resolver las cláusulas o los literales.

la permutación:

- **Permutaciones de cláusulas**: La permutación indica el orden en que se van a resolver las cláusulas. En este caso el tamaño del individuo coincide con el número de cláusulas del problema. Para cada individuo generado por el EDA, se crea una permutación que indica el orden que se va a seguir a la hora de satisfacer las cláusulas.

  Una vez decidido el orden en que se resolverán las cláusulas hay que asignar los valores correspondientes para resolverlas. Para ello es necesario decidir qué criterios se utilizarán para satisfacer cada cláusula. Teniendo en cuenta que cada cláusula contiene k literales y que es suficiente con satisfacer uno de ellos para que se satisfaga la cláusula, se trata de seleccionar en cada caso el literal que va a satisfacer la cláusula y asignarle a esa variable el valor correspondiente. Es importante recordar que un literal es una variable negada o no y por tanto, resolver un literal consiste en asignarle a esa variable el valor correspondiente dependiendo de si está negada o no. En este momento es cuando se puede utilizar información inicial del problema relativa a la actividad de cada variable. Con este objetivo, se llevará la cuenta de en cuantas cláusulas aparece cada variable y con que signo.

  Inicialmente todos los valores de las variables están por decidir y por lo tanto se puede asignar a cualquier variable de la cláusula, el valor que más interesa. Lo importante es elegir la variable que además de resolver esa cláusula, impida la resolución del menor número de cláusulas posible. Esto es, cada vez que haya que resolver una cláusula y existan más de una variable de esa cláusula sin asignar, elegimos aquella variable que esté en menos cláusulas con el valor complementario del que está en esta cláusula. Si esa variable en la cláusula está en positivo, tomará el valor 1 y por tanto el valor no elegido será el 0, de manera que para todas las cláusulas que tengan esa variable en negativo habrá que elegir otra

variable para su resolución.

A medida que se avanza en la resolución de cláusulas, puede ocurrir que todas las variables de la cláusula que corresponde resolver tengan valores asignados, y que ninguno satisfaga la cláusula. En ese caso, marcamos esa cláusula como no satisfecha. La función objetivo lleva la cuenta de todas las cláusulas no satisfechas, por lo tanto, este valor se debe minimizar.

La idea es que el algoritmo evolutivo, durante el proceso de aprendizaje, identifique cuales son las cláusulas más conectadas, que normalmente serán las más difíciles de satisfacer, de manera que estas aparezcan en primer lugar en los individuos de la siguiente generación.

La ventaja de esta representación es que, a la hora de asignar los valores que tomarán las variables del problema SAT, se tienen en cuenta las características propias del problema concreto que se intenta resolver, tales como la actividad de la variable y la conectividad de la cláusula. Se pretende que las cláusulas más conectadas tengan prioridad y que el EDA sea capaz de detectar esta conectividad. De esta manera, se tiene en cuenta la dificultad de la cláusula al establecer el orden en que estas se resuelven.

- **Permutaciones de variables**: En este caso la permutación indica el orden en que se van a resolver cada una de las variables del problema SAT. Esta representación consiste en crear un individuo formado por valores continuos, de tamaño igual al número de variables del problema. A partir de cada individuo generado por el EDA, se crea una permutación que indicará el orden en que se asignan los valores booleanos correspondientes a cada variable.

  Una vez establecido el orden en que se resuelven las variables hay que asignar a estas un valor. A la hora de asignar un valor a una variable, se tiene en cuenta la actividad de la variable (en cuantas cláusulas se encuentra esa variable) y se le asigna el valor que más cláusulas resuelve. A continuación, se vuelve a calcular, para cada variable, en cuantas cláusulas aparece y con que signo, pero ahora sin tener en cuenta aquellas cláusulas que ya están resueltas. El proceso continua con la siguiente variable marcada por la permutación.

  En este caso, también se utiliza información inicial del problema para resolver cada variable. Sin embargo, aquí no se tiene en cuenta la dificultad de las cláusulas.

Optimización local en representación con permutaciones de cláusulas

En la representación con permutaciones de cláusulas se establece el orden en que se resuelven las cláusulas. Añadir optimización local en este caso supone cambiar el orden en que se va a resolver alguna cláusula. Para ello, se puede elegir intercambiar

el orden en que se resuelve alguna cláusula no satisfecha con alguna de las resueltas anteriormente. La selección de cual de las anteriores se intercambia, se puede hacer teniendo en cuenta su probabilidad:

$$P_j = \frac{D(j,k)}{\sum_{i=1}^{k-1} D(i,k)} \tag{5.4}$$

donde k es el orden de la cláusula que no se satisface, j el orden de la cláusula de la cual estamos calculando su probabilidad y $D(j,k)$ es el número de literales que tienen en común las cláusulas j y k. Una vez calculadas la probabilidades de las cláusulas resueltas, anteriores a la cláusula de orden k, para elegir la cláusula a intercambiar se puede utilizar una probabilidad proporcional a la probabilidad calculada.

Adaptación de la función objetivo en representación con permutaciones

En la sección 5.4.5 se ha descrito cómo adaptar la función objetivo de forma dinámica dando mayor peso a las cláusulas que no se resuelven. Esta forma de adaptar la función objetivo puede utilizarse cuando las variables del individuo coinciden con las variables del problema. Sin embargo, cuando utilizamos la representación basada en permutaciones, es necesario pensar en nuevas formas de adaptación de la función objetivo. A continuación se presentan algunas propuestas concretas de cómo adaptar la función objetivo en representaciones de individuos mediante permutaciones.

- **Adaptación de la función objetivo en representación mediante permutaciones de cláusulas**

  Al utilizar representación mediante permutaciones de cláusulas, se busca que sea el proceso de aprendizaje del algoritmo evolutivo el que decida el orden más apropiado en el que se deben de resolver las cláusulas. Normalmente se resolverán primero la cláusulas más conectadas, es decir, aquellas que contengan variables que aparecen en un mayor número de cláusulas. La función objetivo utilizada en este caso cuenta el número de cláusulas no resueltas y el óptimo se alcanza cuando la función objetivo vale 0. Con esta función se ayuda al proceso evolutivo a ir marcando el orden correspondiente. Sin embargo, llega un momento en que existen individuos diferentes con el mismo valor objetivo, lo cual genera ambigüedad en el proceso. Mediante la adaptación de la función objetivo se pretende resolver esta ambigüedad. Una forma de hacerlo puede ser que la función objetivo, además de contar el número de cláusulas sin resolver, incorpore información sobre la conectividad de esas cláusulas no resueltas, de manera que individuos diferentes obtengan valores diferentes. Para ello se propone una nueva función objetivo, $F(\boldsymbol{x})$, que al evaluar cada individuo obtenga un número real en el que la parte entera indique el número de cláusulas sin resolver mientras

que la parte decimal aporte información sobre la conectividad de esas cláusulas que no han sido resueltas:

$$F(\boldsymbol{x}) = NS(\boldsymbol{x}) + W * 10e^{-k} \tag{5.5}$$

donde $NS(\boldsymbol{x})$ es el número de cláusulas no satisfechas y W representa el valor calculado en base a la conectividad de esas cláusulas. Esto puede hacerse sumando, por cada cláusula no resuelta, el número de cláusulas en las que aparecen todas las variables de esa cláusula. Para que este valor esté formado por el número de cláusulas no resueltas como parte entera y W como parte decimal, se multiplica W por $10e^{-k}$, donde k depende del número de cláusulas del problema.

- **Adaptación de la función objetivo en representación mediante permutaciones de variables**

  En la representación mediante permutaciones de variables, el individuo indica el orden en que se resolverán cada una de las variables booleanas del problema. Igual que en el caso anterior, la función objetivo cuenta el número de cláusulas no resueltas, presentando ambigüedad. En este caso se propone adaptar la función objetivo añadiendo información sobre la actividad de las variables que aparecen en cláusulas no resueltas. Para ello se puede utilizar la misma fórmula anterior:

$$F(\boldsymbol{x}) = NS(\boldsymbol{x}) + W * 10e^{-k} \tag{5.6}$$

  donde $NS(\boldsymbol{x})$ sigue siendo el número de cláusulas no satisfechas, aunque ahora W puede tener un significado diferente. A continuación se proponen tres criterios diferentes a la hora de calcular W. En los dos primeros casos –WA y WB– se tiene en cuenta la actividad de las variables, aunque de forma diferente, mientras que en el último caso –WC–, se tienen en cuenta las cláusulas no resueltas.

  - Una primera opción, que identificaremos como WA, consiste en calcular W multiplicando los pesos de las variables de cada cláusula no resuelta, considerando como peso de cada variable el número de cláusulas en las que aparece dicha variable.

$$Weight(X_i) = TotClauses(X_i) \tag{5.7}$$

$$WA = \prod_{i=1}^{v} Weight(X_i) \tag{5.8}$$

    donde $TotClauses(X_i)$ indica el número de cláusulas que contienen a la variable X_i, con $i = 1, \ldots, v$ y v el número de variables del problema SAT.

- Una segunda opción, identificada como WB, consiste en calcular W como la suma de los restos de las variables de la cláusula, entendiendo por resto ($Resto$) el número de variables que todavía no se han resuelto.

$$WB = \sum(Resto) \tag{5.9}$$

- La tercera opción que se plantea y que denominaremos WC, consiste en asignar un peso a cada cláusula de forma dinámica, dependiendo de si esta se resuelve o no. Inicialmente se le asigna a cada cláusula un peso $w_i = 1$, con $i = 1 \dots u$, siendo u el número de cláusulas. Posteriormente, se van actualizando los pesos de las cláusulas dependiendo de si estas se resuelven o no. Esta actualización consiste en mantener el valor del peso de las cláusulas no resueltas y actualizar el peso de las cláusulas resueltas de manera que tengan un peso menor, ya que el objetivo es minimizar $F(\boldsymbol{x})$. De esta manera, el valor de W se calcula como la suma de los pesos de todas las cláusulas.

$$WC = \sum_{i}^{u}(w_i) \tag{5.10}$$

5.6 Hibridación de EDAs con otros algoritmos de optimización local

Con el termino hibridación nos referimos a la combinación de más de un algoritmo con el objetivo de aprovechar las ventajas de cada uno y aumentar el rendimiento de ambos. A la hora de buscar estrategias para resolver SAT, encontramos ejemplos donde se han combinado el enfoque de búsqueda local con algoritmos completos basados en el método DPLL (Ferris y Froehlich, 2004; Fang y Hsiao, 2007; Letombe y Marques-Silva, 2008; Wang y col., 2008) o con algoritmos evolutivos (Pelikan y Goldberg, 2003; Hao y col., 2003; Lardeux y col., 2006). En este caso se plantea la combinación de un algoritmo EDA con otro algoritmo de búsqueda local para resolver instancias más complejas de problemas SAT.

En general, un algoritmo de búsqueda local comienza con una asignación de las variables que se puede obtener de forma aleatoria y va realizando cambios en los valores de las variables de manera que se mantengan aquellos cambios que aumenten el total de cláusulas satisfechas. Si una vez realizado un determinado número de cambios (MAX-FLIPS), no se consigue satisfacer todas las cláusulas, el algoritmo realiza un nuevo intento con una nueva inicialización. Esto se repite hasta un determinado número de intentos (MAX-TRIES) o hasta encontrar el óptimo. Normalmente, las diferencias más significativas entre unos algoritmos de búsqueda local y otros están en la forma en que seleccionan la variable a la que se va a cambiar su valor.

Según esta descripción, encontramos que la hibridación de un EDAs con un algoritmo de búsqueda local puede realizarse de diferentes formas, algunas de las cuales describimos a continuación.

- **EDA + algoritmo de búsqueda local:** Consiste en ejecutar un algoritmo EDA durante un determinado número de generaciones y utilizar el mejor individuo encontrado por el EDA como asignación inicial para ejecutar el algoritmo de búsqueda local. El problema de esta solución es que todos los intentos que realiza el algoritmo de búsqueda local comienzan con la misma asignación inicial (el mejor individuo encontrado por el EDA). Esto se puede mejorar utilizando los n mejores individuos encontrados por el EDA de manera que cada uno de ellos se utilice como asignación inicial en cada intento del algoritmo de búsqueda local.

- **realimentación EDA-algoritmo de búsqueda local:** En esta opción se van ejecutando alternativamente el EDA y el algoritmo de búsqueda local. En cada generación, el EDA proporciona al algoritmo de búsqueda local el mejor individuo encontrado para que este lo utilice como asignación inicial. Por su parte, el algoritmo de búsqueda local, proporciona al EDA la mejor asignación encontrada, de manera que se incluya como un nuevo individuo en la población que se utilizará en la siguiente generación. Esto se puede realizar con un solo individuo o con los n mejores.

- **algoritmo de búsqueda local + EDA:** También podemos comenzar ejecutando un algoritmo de búsqueda local tantas veces como sea necesario para generar tantas soluciones como individuos utilice el algoritmo EDA. En este caso es el EDA el que se ejecuta con una población inicial generada con las asignaciones encontradas en las diferentes ejecuciones del algoritmo de búsqueda local.

5.7 Resultados experimentales

En esta sección se presentan los experimentos realizados para comprobar el rendimiento, tanto de los EDAs como de los EBCOAs, al utilizar cada una de las opciones comentadas en las secciones anteriores para la resolución de problemas SAT. También se comentan las conclusiones extraídas de los resultados obtenidos en estos experimentos.

En todos los experimentos que se describen a continuación se han utilizado instancias *3*-SAT elegidas al azar de la colección de problemas ofrecidos por SATLIB[3]. Los

[3]Holger H. Hoos y Thomas Stützle: SATLIB: An online resource for research on SAT. In: I.P.Gent, H.v.Maaren, T.Walsh, editors, SAT 2000, pp.283-292, IOS Press, 2000. SATLIB está disponible online en www.satlib.org.

ficheros elegidos contienen definiciones de problemas SAT para los cuales se puede encontrar una solución satisfactible. Es una práctica común utilizar la proporción entre el número de variables del problema y el número de cláusulas para definir la complejidad de un problema SAT. En todos estos experimentos se han utilizado instancias con el ratio $u/v \approx 4,3$, siendo u el número de cláusulas y v es el número de variables ya que, como se ha demostrado en (Mitchell y col., 1992), representan los casos más difíciles.

Comenzaremos comprobando el comportamiento de algunos EDAs y EBCOAs (ver capítulo 4) discretos cuando utilizamos las representaciones con valores discretos en las variables del individuo presentadas en la sección 5.4. Posteriormente comprobaremos la eficiencia de EDAs y EBCOAs continuos al utilizar las representaciones mediante permutaciones descritas en la sección 5.5. Por último se presentan los experimentos realizados combinando EDAs y EBCOAs con otros algoritmos de optimización local según se ha comentado en la sección 5.6.

5.7.1 Experimentos realizados utilizando diferentes representaciones discretas

El objetivo de estos experimentos consiste en comprobar el comportamiento de EDAs y EBCOAs con las tres representaciones que utilizan valores discretos en las variables del individuo presentadas en la sección 5.4. Para cada una de ellas se han utilizado 5 instancias diferentes de problemas SAT con 20 variables y 91 cláusulas. Se han realizado 10 ejecuciones con cada instancia y con cada uno de los diferentes algoritmos.

En el caso de los EDAs, se ha seleccionado un representante de cada una de las categorías de EDAs, según el número de dependencias entre variables que tienen en cuenta. Concretamente los algoritmos $UMDA$, $MIMIC$ y $EBNA$ descritos en la sección 3.2. En estos casos se ha utilizado un tamaño de población de 1000 individuos de los que se seleccionan los 500 mejores en cada generación.

Para los experimentos con EBCOAs se han utilizado los diferentes algoritmos definidos en la sección 4.3 (*Naïve Bayes*, *Selective Naïve Bayes*, $FSSJ$, $BSEJ$ y TAN). El tamaño de población en las ejecuciones es de 100 individuos ($R = 100$), que se han dividido en 3 clases de igual tamaño ($|K| = 3$) de las cuales se han seleccionado dos ($|C| = 2$), los mejores y los peores, para construir el clasificador. Como ya comentamos en el capítulo dedicado a los EBCOAs, concretamente en la sección 4.4.4, el tamaño de la población es un parámetro que tiene mucha importancia en el rendimiento de los EBCOAs. Antes de decidir el valor de este parámetro, se realizaron diferentes pruebas para elegir el que mejores resultados obtenía. El criterio de parada utilizado en todos los casos ha sido encontrar una solución satisfactoria o realizar un máximo de 500 generaciones. Cada algoritmo se ha ejecutado utilizando las tres representaciones de individuos con valores discretos definidas en la sección 5.4.

La Tabla 5.1 muestra los resultados de estos experimentos. La columna % indica el porcentaje de ejecuciones en las que se ha encontrado una solución satisfactible. La

	Bits			**Path**		
	%	*Dif.*	*Eval.*	*%*	*Dif.*	*Eval.*
EDA_{UMDA}	100	0	6099.26	18	1.10	829066
EDA_{MIMIC}	100	0	7045.72	18	1.10	846967
EDA_{EBNA}	100	0	5637.60	18	1.98	838259
$EBCOA_{NaiveBayes}$	86	0.18	2222.84	6	1.72	54443.32
$EBCOA_{SelectiveNB}$	82	0.20	2256.10	2	1.78	53788.66
$EBCOA_{FSSJ}$	98	0.02	2472.10	0	2.76	50253.64
$EBCOA_{BSEJ}$	92	0.08	1700.60	2	1.74	52330.42
$EBCOA_{TAN}$	52	0.66	5285.12	0	4.60	52767.50

	Cláusula		
	%	*Dif.*	*Eval.*
EDA_{UMDA}	16	1.28	57875.76
EDA_{MIMIC}	16	1.30	65456.42
EDA_{EBNA}	18	1.28	72373.1
$EBCOA_{NaiveBayes}$	0	39.12	49832.62
$EBCOA_{SelectiveNB}$	0	41.78	49856.22
$EBCOA_{FSSJ}$	0	53.56	49753.52
$EBCOA_{BSEJ}$	0	49.50	49881.02
$EBCOA_{TAN}$	0	26.02	49639.46

Tabla 5.1: Resultados obtenidos en 10 ejecuciones realizadas con cada algoritmo, con 5 problemas SAT diferentes, cada uno de ellos con 20 variables y 91 cláusulas.

columna *Dif.* representa la media del número de cláusulas que quedaban por resolver en la última generación y la columna *Eval.* representa el número de evaluaciones realizada, es decir, el número de individuos diferentes que han sido analizados durante el proceso de búsqueda. Se ha representado como *Bits* los casos en los que se ha utilizado representación mediante cadena de bits, como *Path* la utilización de representación mediante el camino y como *Cláusula* aquellos casos en que la representación ha sido utilizando la cláusula.

Como puede verse en estos resultados, la representación mediante cadena de bits es la que muestra un mejor rendimiento. Los EDAs consiguen resolver los problemas SAT de 20 variables en todos los casos en los que la representación utilizada es mediante cadena de bits. En cuanto al comportamiento de los diferentes clasificadores utilizados por los EBCOAs con este tipo de representación, puede comprobarse que cuando se utilizan cualquiera de las dos opciones del clasificador *Semi Naïve Bayes*, se obtiene una solución satisfactible en más del 90 % de las ejecuciones. Concretamente con el clasificador $FSSJ$ se obtiene el óptimo en el 98 % de los casos. Sin embargo, al utilizar el clasificador TAN el porcentaje de ejecuciones que encuentran una solución satisfactible es sólo del 50 %.

Los otros dos tipos de representaciones obtienen resultados considerablemente peores debido a que exploran un espacio de búsqueda mayor y un mayor número de individuos diferentes con el mismo valor fitness. En la literatura encontramos diferentes trabajos que llegan a la misma conclusión con otros algoritmos evolutivos como los genéticos (Gottlieb y col., 2002).

Resultados obtenidos al aplicar optimización local en representaciones discretas

En los experimentos realizados anteriormente, los EDAs alcanzan el óptimo en todas las ejecuciones en las que se ha utilizado representación con cadena de bits. Sin embargo, en el caso de los EBCOAs, no siempre se alcanza el óptimo. Hemos realizado estos mismos experimentos con EBCOAs, pero ahora añadimos la optimización local descrita en la sección 5.4.5, para estos tipos de representaciones con valores discretos. En este caso no se ha aplicado a EDAs ya que, para este tamaño de problema, alcanzan el óptimo sin necesidad de aplicar técnicas adicionales. Más adelante se aplicará a problemas de mayor tamaño. La tabla 5.2 muestra los resultados obtenidos en estos experimentos.

Al aplicar optimización local se alcanza el óptimo en todos los casos en los que se utiliza representación mediante cadena de bits. Sin embargo, es importante destacar que, en este caso, aumenta considerable el número de individuos evaluados en relación a cuando no se utilizaba optimización local. Esto demuestra que la incorporación de optimización local profundiza en zonas concretas del espacio de búsqueda y ayuda a encontrar el óptimo.

Resultados obtenidos al añadir adaptación de la función objetivo mediante pesos en representaciones discretas

Estos mismos experimentos se han repetido adaptando la función objetivo mediante pesos tal y como se ha explicado en la sección 5.4.5. Igual que en caso anterior, se aplica a EBCOAs pero no a EDAs. La tabla 5.3 muestra los resultados obtenidos en este caso.

Al añadir este tipo de adaptación de la función objetivo, se obtiene una solución satisfactible en el 100 % de los casos sólo cuando utilizamos el clasificador *Selective Naïve Bayes* y siempre que se utilice representación mediante cadena de bits. Para el resto de EBCOAs la tasa de éxito también es más alta que cuando no se aplica ningún tipo de técnica adicional (del 98 % para *Naïve Bayes* y $FSSJ$), excepto para TAN que sigue siendo de un 72 %. En la representación mediante path, también mejora el porcentaje de casos en los que alcanza el óptimo cuando utilizamos *Naïve Bayes*, *Selective Naïve Bayes* y $BSEJ$. En estos casos el porcentaje de ejecuciones que encuentra una solución satisfactible supera al 50 % y en el resto de los casos se queda a falta de

	Bits			Path		
	%	*Dif.*	*Eval.*	*%*	*Dif.*	*Eval.*
$\text{EBCOA}_{NaiveBayes}$	100	0.00	7333.28	16	1.40	51113.16
$\text{EBCOA}_{SelectiveNB}$	100	0.00	7277.98	12	1.48	50822.18
EBCOA_{FSSJ}	100	0.00	7363.50	0	2.20	49956.84
EBCOA_{BSEJ}	100	0.00	7339.70	6	2.02	48897.10
EBCOA_{TAN}	100	0.00	7277.98	0	4.10	50130.92

	Cláusula		
	%	*Dif.*	*Eval.*
$\text{EBCOA}_{NaiveBayes}$	0	12.52	49656.04
$\text{EBCOA}_{SelectiveNB}$	0	12.90	49654.14
EBCOA_{FSSJ}	0	29.28	49655.04
EBCOA_{BSEJ}	0	15.04	49600.00
EBCOA_{TAN}	0	70.44	49625.38

Tabla 5.2: Resultados obtenidos al aplicar optimización local en 10 ejecuciones con 5 problemas SAT diferentes, cada uno con 20 variables y 91 cláusulas.

resolver una o dos cláusulas. En cualquier caso, los resultados siguen siendo mejores al utilizar representación mediante secuencia de bits.

Al comparar EDAs y EBCOAS, los EDAs obtienen un mejor rendimiento en este caso concreto ya que, con la representación mediante cadena de bits, alcanzan el óptimo en todos los casos sin necesidad de aplicar ninguna técnica adicional. Respecto a las técnicas adicionales que se han utilizado con los EBCOAs, en ambos casos mejoran el rendimiento ayudando a encontrar el óptimo. Sin embargo, se muestra más eficaz la optimización local que la adaptación de la función objetivo.

Rendimiento de los EDAs y EBCOAs al aumentar el tamaño del problema

Los experimentos anteriores se han realizado con problemas SAT de 20 variables. Para comprobar si el comportamiento de estos algoritmos se mantiene al aumentar el tamaño del problema, y por tanto el espacio de búsqueda, se han repetido estos experimentos con problemas SAT de mayor tamaño, manteniendo el ratio $u/v \approx 4,3$ entre el número de cláusulas y el número de variables. Para ello, se han utilizado nuevos problemas SAT de la librería SATLIB (Hoos y Stützle, 2000) con 50 variables y 218 cláusulas. Igual que en el caso anterior, se han utilizado 5 instancias diferentes y se han ejecutado 10 veces cada uno de los algoritmos con cada instancia. El resto de parámetros utilizados han sido los mismos que en el caso anterior con 20 variables, excepto el tamaño de población en los EBCOAs que al aumentar el tamaño del individuo, se

	Bits			Path		
	%	*Dif.*	*Eval.*	*%*	*Dif.*	*Eval.*
EBCOA$_{NaiveBayes}$	98	0.04	2155.10	58	0.76	31726.54
EBCOA$_{SelectiveNB}$	100	0.00	4912.80	52	0.88	34284.70
EBCOA$_{FSSJ}$	98	0.04	3105.50	0	5.86	49600.00
EBCOA$_{BSEJ}$	94	0.08	5420.42	64	0.76	32599.72
EBCOA$_{TAN}$	72	0.38	14698.92	0	6.42	49600.00

	Cláusula		
	%	*Dif.*	*Eval.*
EBCOA$_{NaiveBayes}$	0	12.58	49600.00
EBCOA$_{SelectiveNB}$	0	11.42	49600.00
EBCOA$_{FSSJ}$	0	33.86	49600.00
EBCOA$_{BSEJ}$	0	12.38	49600.00
EBCOA$_{TAN}$	0	47.86	49600.00

Tabla 5.3: Resultados obtenidos al añadir adaptación de la función objetivo mediante pesos en 10 ejecuciones con 5 problemas SAT diferentes, cada uno con 20 variables y 91 cláusulas. No se ha utilizado optimización local.

	%	*Dif.*	*Eval.*
EDA$_{UMDA}$	36	1.23	25604.44
EDA$_{MIMIC}$	32	1.21	26238.1
EDA$_{EBNA}$	56	0.69	24032
EBCOA$_{NaiveBayes}$	18	1.62	15933.72
EBCOA$_{SelectiveNB}$	8	1.78	16633.78
EBCOA$_{FSSJ}$	34	1.14	98698.86
EBCOA$_{BSEJ}$	14	1.90	17962.96
EBCOA$_{TAN}$	0	4.06	25691.80

Tabla 5.4: Resultados obtenidos en 10 ejecuciones realizadas con cada algoritmo, con 5 problemas SAT diferentes, cada uno de ellos con 50 variables y 218 cláusulas. Se ha utilizado representación mediante cadena de bits.

ha ampliado a 250 ($R = 250$). Teniendo en cuenta las conclusiones anteriores, esta vez solo se ha utilizado la representación mediante cadena de bits.

La tabla 5.4 muestra los resultados obtenidos al ejecutar los diferentes algoritmos sin aplicar ningún tipo de técnica adicional. Como se puede ver en esta tabla, se mantiene la tendencia observada en los experimentos anteriores en cuanto al comportamiento de cada algoritmo, aunque al aumentar la complejidad del problema, esto se traduce en un rendimiento más bajo que cuando trabajábamos con 20 variables. En este caso, el clasificador *Semi Naïve Bayes* en la versión hacia delante ($FSSJ$),

	%	*Dif.*	*Eval.*
EDA_{UMDA}	100	0.00	73242.90
EDA_{MIMIC}	100	0.00	73408.68
EDA_{EBNA}	100	0.00	73261.20
$\text{EBCOA}_{NaiveBayes}$	100	0.00	98058.90
$\text{EBCOA}_{SelectiveNB}$	100	0.00	93900.54
EBCOA_{FSSJ}	100	0.00	75334.26
EBCOA_{BSEJ}	100	0.00	84792.22
EBCOA_{TAN}	100	0.00	274786.08

Tabla 5.5: Resultados obtenidos al aplicar optimización local en 10 ejecuciones con 5 problemas SAT diferentes, cada uno con 50 variables y 218 cláusulas. Se ha utilizado representación mediante cadena de bits.

	%	*Dif.*	*Eval.*
EDA_{UMDA}	100	0.00	3886.66
EDA_{MIMIC}	100	0.00	4357.40
EDA_{EBNA}	100	0.00	4305.48
$\text{EBCOA}_{NaiveBayes}$	50	1.06	69199.38
$\text{EBCOA}_{SelectiveNB}$	52	0.94	66921.32
EBCOA_{FSSJ}	62	0.74	72924.72
EBCOA_{BSEJ}	46	1.14	71453.36
EBCOA_{TAN}	10	2.82	110144.16

Tabla 5.6: Resultados obtenidos al añadir adaptación de la función objetivo mediante pesos en 10 ejecuciones con 5 problemas SAT diferentes, cada uno con 50 variables y 218 cláusulas. Se ha utilizado representación mediante cadena de bits.

obtiene resultados comparables a los obtenidos por los diferentes EDAs.

En la tabla 5.5 se muestran los resultados obtenidos al realizar estos mismos experimentos añadiendo optimización local. Mediante la optimización local se consigue encontrar una solución satisfactible en todos los casos a costa de aumentar considerablemente el número de evaluaciones realizadas. Esto confirma la eficiencia de esta técnica.

También se obtienen buenos resultados con los EDAs al añadir adaptación de la función objetivo mediante pesos. Estos resultados pueden verse en la tabla 5.6. En este caso, todos los algoritmos mejoran su rendimiento, pero mientras los EBCOAs obtienen una solución satisfactible en alrededor del 50 % de la ejecuciones, (algo más en el caso del *FSSJ*), los EDAs consiguen encontrar el óptimo siempre y con menos evaluaciones.

5.7.2 Experimentos realizados utilizando permutaciones

Para realizar los experimentos utilizando permutaciones se han utilizado algoritmos que trabajan con variables continuas. Estos experimentos se han realizado utilizando EDAs continuos pertenecientes a diferentes categorías, según el número de dependencias entre variables que tienen en cuenta. Concretamente $UMDA_c$, $MIMIC_c^G$ y $EGNA_{ee}$. En estos casos se ha utilizado un tamaño de población de 2000 individuos de los que se han seleccionado los 1000 mejores en cada generación. El criterio de parada ha sido encontrar el óptimo –resolver todas las cláusulas– o realizar un máximo de 100 generaciones. Igual que en el caso de los discretos, las pruebas se han realizado con problemas SAT de SATLIB (Hoos y Stützle, 2000), manteniendo un ratio $u/v \approx 4,3$. Para cada experimento se han utilizado 5 instancias de problemas SAT y con cada una de ellas se han realizado 10 ejecuciones.

Los experimentos con EBCOAs se han realizado con las versiones continuas de estos algoritmos presentadas en el capítulo 4. En este caso, se han utilizado diferentes tamaños de población, buscando en cada caso el que obtuviera mejores resultados. El criterio de parada es el mismo que con EDAs.

Los resultados obtenidos en cada uno de los casos se comentan a continuación.

Resultados obtenidos al utilizar representación mediante permutaciones de cláusulas

Para problemas SAT con un tamaño de 20 variables y 91 cláusulas, los EDAs utilizados, consiguen satisfacer todas las cláusulas, ya en la primera generación, cuando se utiliza representación mediante permutaciones de cláusulas.

Para ver la eficiencia de los EDAs para este tipo de representación es necesario probarlo con problemas de mayor tamaño. Con problemas con 50 variables y 218 cláusulas se consigue encontrar una solución satisfactible en algo más del 60 % de las ejecuciones ($MIMIC_c^G$ encuentra el óptimo en el 90 % de los casos). Estos resultados se pueden ver en la tabla 5.7, donde la primera columna –v– indica el número de variables del problema SAT.

Al ampliar el tamaño del problema a 100 variables y 430 cláusulas, se ha dejado que cada ejecución realice un máximo de 300 generaciones. En estos casos, no se encuentra una solución satisfactible en ninguna de las ejecuciones. Ya en la generación 100, se obtienen resultados donde quedan 2 ó 3 cláusulas sin resolver. Sin embargo, al realizar más generaciones no consigue mejorar y termina con 2 ó 3 cláusulas sin satisfacer.

Al repetir estos experimentos, pero ahora utilizando tanto optimización local como adaptación de la función objetivo (sección 5.4.5) con problemas de 20 y 50 variables, el problema se resuelve en todos los casos en la generación 0. Sin embargo, al intentarlo con problemas de 100 variables acaba sin encontrar el óptimo en ningún caso.

Todos estos resultados confirman que la representación mediante permutaciones, en este caso, no mejora el rendimiento de los EDAs, ya que estos algoritmos no pueden

		Permutaciones de cláusulas			*Permutaciones de variables*		
	v	*%*	*Dif.*	*Eval.*	*%*	*Dif.*	*Eval.*
	20	100	0.00	1119,88	100	0.00	1000
$UMDA_c$	50	62	0.72	197104	100	0.00	4756.24
	20	100	0.00	1059.94	100	0.00	1000
$MIMIC_c^G$	50	90	0.27	68114.6	100	0.00	4876.12
	20	100	0.00	1000	100	0.00	1000
$EGNA_{ee}$	50	66	0.56	16142.1	100	0.00	1000

Tabla 5.7: Resultados obtenidos al utilizar permutaciones de cláusulas y de variables para resolver problemas SAT de 20 y 50 variables.

encontrar relaciones entre las variables del problema con este tipo de representación. Podríamos decir que los resultados que se obtienen son debidos a que a la hora de asignar los valores a cada variable se trabaja con información extraída del problema original.

Resultados obtenidos al utilizar representación mediante permutaciones de variables

En los experimentos realizados con EDAs utilizando permutaciones de variables, se alcanza el óptimo en todas las ejecuciones realizadas con problemas de 20 y 50 variables (tabla 5.7). Al ampliar a 100 el número de variables ocurre lo mismo que en el caso anterior con permutaciones de cláusulas.

Las diferentes adaptaciones de la función objetivo que hemos comentado en la sección 5.5 las hemos aplicado a problemas de 100 variables, ya que con tamaños más pequeños de problemas se alcanza el óptimo siempre en la primera generación. Los resultados han sido los mismos que cuando no hemos aplicado esta técnica. Termina las 300 generaciones sin encontrar una solución satisfactoria en ninguna de las ejecuciones.

También se han realizado experimentos para comprobar el comportamiento de los EBCOAs con este tipo de representación y ocurre lo mismo que en las ejecuciones con EDAs.

La conclusión a la que podemos llegar, a la vista de estos resultados, es que en este tipo de representaciones resulta muy importante la utilización de la información obtenida de la definición del problema. Debemos recordar que el individuo marca el orden en que se resolverán cada una de las variables, bien marcada por la cláusula a la que pertenece en las permutaciones de cláusulas o directamente en las permutaciones de variables. Luego, para decidir el valor que se asigna a cada variable, se tiene en cuenta en cuantas cláusulas aparece esa variable y con qué valor. Es decir, se utilizan los conceptos de conectividad de la cláusula y actividad de la variable descritos en la sección 5.2. La utilización de esta información es la que hace que ya en la primera generación se obtengan buenos resultados y, en la mayoría de los casos, se llegue

a resolver el problema. Sin embargo, en los casos en que no resuelve el problema en esta primera generación, luego apenas mejora. Esto es debido a la ambigüedad que presenta la función objetivo, ya que hay individuos muy diferentes que obtienen el mismo valor fitness. Es decir, hay individuos que dejan n cláusulas sin resolver (donde n puede ser un valor muy bajo: de 1 a 4 aproximadamente) pero la función objetivo no establece ninguna diferencia entre ellos. También hay que tener en cuenta el proceso de aprendizaje de los EDAs. Durante el aprendizaje, los EDAs buscan las interdependencias entre las variables del individuo y las representan en el modelo gráfico probabilístico que se utilizará posteriormente para crear nuevos individuos. En este caso, las variables del individuo establecen el orden en que se resuelven las variables del problema, de manera que las dependencias entre estas variables no representan características del problema.

Todo ello confirma que la utilización de permutaciones no aumenta el rendimiento de estos algoritmos, ya que el proceso de aprendizaje no puede encontrar interdependencias entre variables del problema original al trabajar con variables que representan un orden y no las variables del problema.

5.7.3 Experimentos con hibridaciones

En los experimentos con hibridaciones hemos combinado EDAS y EBCOAs con los algoritmos de búsqueda local GSAT (Selman y col., 1992) y WalkSAT (Selman y col., 1994), ya que son algoritmos que han demostrado obtener buenos resultados en la resolución de instancias SAT (Kautz y col., 2009; Letombe y Marques-Silva, 2008).

GSAT comienza creando una solución de manera aleatoria para, a continuación, ir realizando cambios en los valores de las variables, buscando mejorar la solución inicial. Cuando ha realizado un determinado número de cambios ($MAX - FLIPS$) sin encontrar el óptimo, vuelve a intentarlo con una nueva solución aleatoria. Esto se repite hasta encontrar el óptimo o hasta realizar un máximo número de intentos ($MAX - TRIES$). La figura 5.7 muestra el pseudocódigo de este algoritmo.

WalkSAT sigue el mismo esquema que GSAT pero seleccionando siempre la variable cuyo valor va a cambiar, de entre las variables de una cláusula no satisfecha. Si hay una variable en C, siendo C una cláusula no satisfecha elegida al azar, que al cambiar su valor no provoca que ninguna de las cláusulas satisfechas actualmente pase a insatisfecha, se cambia el valor de esa variable. De lo contrario, con una cierta probabilidad p, se cambia el valor de una variable en C de forma aleatoria, y con probabilidad $1 - p$, se cambia una variable en C que minimice el número de cláusulas satisfechas actualmente. El pseudocódigo de este algoritmo se muestra en la figura 5.8.

En los experimentos con hibridaciones elegimos problemas SAT suficientemente grandes para que no sean directamente resolubles por el algoritmo de búsqueda local que utilizamos en la hibridación. De esta manera podemos comprobar la colaboración de ambos algoritmos, el EDA y el de búsqueda local, en la resolución de problemas más complejos. En este caso se han utilizado problemas de 250 variables y 1075 cláusulas,

GSAT

Repetir Para $i = 1, \ldots, MAX - TRIES$
  $T \leftarrow$ Generar una asignación al azar
    Repetir Para $j = 1, \ldots, MAX - FLIPS$
      Si T satisface el conjunto de cláusulas CL *entonces devolver* T;
      $v \leftarrow$ Seleccionar una variable proposicional tal que un cambio en el valor asignado a dicha variable, incremente el número total de cláusulas de CL satisfechas por T
      $T \leftarrow T$ con la asignación de v cambiada
devolver: "no se ha encontrado una asignación que satisfaga CL";

Figura 5.7: Pseudocódigo del algoritmo *GSAT*.

WalkSAT

Repetir Para $i = 1, \ldots, MAX - TRIES$
  $T \leftarrow$ Generar una asignación al azar
    Repetir Para $j = 1, \ldots, MAX - FLIPS$
      Si T satisface el conjunto de cláusulas CL *entonces devolver* T;
      $C \leftarrow$ una cláusula no satisfecha elegida al azar
      Si $\exists x \in C$ que no rompe ninguna cláusula
        entonces $v \leftarrow x$;
        si no
        con probabilidad p :
        $v \leftarrow$ una variable en C elegida al azar;
        con probabilidad $1 - p$:
        $v \leftarrow$ una variable en C que rompa el menor número de cláusulas;
      $T \leftarrow T$ con la asignación de v cambiada
devolver: "no se ha encontrado una asignación que satisfaga CL";

Figura 5.8: Pseudocódigo del algoritmo *WalkSAT*.

500 variables y 2150 cláusulas y 750 variables y 3225 cláusulas que han sido generados utilizando el generador *mkcnf* desarrollado por Allen Van Gelder[4]. Para todas las instancias de problemas SAT utilizadas existe una solución que satisface todas las cláusulas. Además, en todos los casos se mantiene el ratio $u/v \approx 4,3$ entre el número de cláusulas y el número de variables. Se han utilizado 30 instancias diferentes con cada

[4]El generador *mkcnf* se encuentra disponible online en http://users.soe.ucsc.edu/ avg/software.html

tamaño de problema y con cada instancia se ha realizado 30 ejecuciones. A continuación se comentan los experimentos realizados utilizando las tres opciones planteadas en la sección 5.6, así como los resultados obtenidos.

Resultados obtenidos al utilizar hibridaciones de EDA + algoritmo de búsqueda local

En los primeros experimentos realizados con hibridaciones se ha utilizado la opción descrita como *EDA + algoritmo de búsqueda local* en la sección 5.6. En esta opción, se comienza ejecutando un EDA durante un determinado número de generaciones (en los experimentos se utilizaron 10 generaciones) y el mejor de los individuos obtenido en esta ejecución se utiliza como inicialización en GSAT. Esta combinación se ha realizado con diferentes instancias de SAT y con diferentes parámetros de los EDAs. Los resultados obtenidos se han comparado con los resultados obtenidos al ejecutar GSAT comenzando con una asignación aleatoria. En todos los casos, se obtienen mejores resultados ejecutando GSAT en solitario, dejando que cada intento comience con una asignación aleatoria, que inicializando GSAT con el mejor de los individuos obtenidos por el algoritmo EDA. También se realizaron estos experimentos utilizando los mejores individuos obtenidos por el EDA, de manera que GSAT empiece cada intento con una inicialización diferente. Tampoco en este caso se han conseguido mejores resultados. El siguiente intento ha sido realizar una hibridación donde se realice una realimentación del EDA con el algoritmo de búsqueda local (el caso comentado como *realimentación EDA - algoritmo de búsqueda local* en la sección 5.6).

Resultados obtenidos al utilizar hibridaciones con realimentación EDA - algoritmo de búsqueda local

Para estos experimentos se ha utilizado el algortimo $UMDA$ (sección 3.2.1) con un tamaño de población de 2000 individuos de los que se seleccionaron los 1000 mejores en cada generación. También se ha realizado un experimento combinando EBCOA (*Naïve Bayes*) (sección 4.2.1) con GSAT y WalkSAT. El tamaño de población utilizado en este caso, ha sido de 2000 individuos ($R = 2000$), que se han dividido en 3 clases de igual tamaño ($|K| = 3$) de las cuales se han seleccionado dos ($|C| = 2$), los mejores y los peores, para construir el clasificador. Se ha realizado un máximo de 10 generaciones de cada algoritmo EDA o EBCOA

La tabla 5.8 muestras los resultados de estos experimentos en los que se han realizado un máximo de 10 generaciones y en cada generación se ha realizado la realimentación de los correspondientes algoritmos. También se han ejecutado los algoritmos GSAT y WalkSAT sin hibridación con el propósito de comparar los resultados. La columna % indica el porcentaje de ejecuciones en las que se ha encontrado una solución satisfactible y la columna *Eval.* representa el número de individuos diferentes que han sido

	v=250, u=1075		*v=500, u=2150*		*v=750, u=3225*	
	%	*Eval.*	*%*	*Eval.*	*%*	*Eval.*
$EBCOA_{NB}$+ GSAT	6,78	4.703.083,61	0,44	5.005.032,54	0,00	5.022.040,00
EDA_{UMDA}+ GSAT	6,89	4.698.135,36	1,12	4.984.865,77	0,00	5.022.040,00
GSAT	18,22	4.368.443,28	1,67	4.956.528,96	0,00	5.000.000,00
$EBCOA_{NB}$+ WalkSAT	26,67	3.707.213,61	25,11	4.253.719,24	15,00	4.774.937,33
EDA_{UMDA}+ WalkSAT	26,67	3.696.350,01	25,44	3.941.799,17	16,89	4.341.644,13
WalkSAT	26,67	3.680.339,91	25,56	3.932.297,70	17,89	4.330.706,38

Tabla 5.8: Resultados obtenidos al realizar experimentos combinando diferentes algoritmos.

analizados durante el proceso de búsqueda en el algoritmo evolutivo correspondiente (EDA o EBCOA) y el número de cambios (flips) realizados en GSAT y WalkSAT.

En el caso de las hibridaciones con GSAT, el porcentaje de ejecuciones en las que se alcanza una solución satisfactible es menor que cuando se ejecuta GSAT sin otro algoritmo. Con WalkSAT sin embargo, este porcentaje es equivalente se combine o no con otros algoritmos.

Resultados obtenidos al utilizar hibridaciones algoritmo de búsqueda local +EDA

En los experimentos realizados utilizando hibridaciones de un *algoritmo de búsqueda local + EDA* la idea es crear, utilizando el algoritmo de búsqueda local, una población inicial para el EDA correspondiente, de manera que la población inicial del EDA no sea una población generada de manera aleatoria. En este caso se ha utilizado WalkSAT como algoritmo de búsqueda local. Por lo tanto, se comienza ejecutando WalkSAT un número suficientes de veces como para que genere N soluciones. En este caso, si el problema SAT se resuelve directamente por WalkSAT, no es posible comprobar el funcionamiento del EDA. Esto obliga a utilizar problemas suficientemente complejos como para que WalkSAT no sea capaz de resolverlos. Concretamente hemos utilizado instancias de 750 variables de entre las utilizadas anteriormente. Las N soluciones generadas por WalkSAT (se ha fijado N=2000), se utilizan como población inicial para el EDA. La condición de parada para el EDA ha sido encontrar el óptimo o realizar un máximo de 30 generaciones.

En ninguno de los experimentos realizados se consiguió encontrar una solución que resolviera todas las cláusulas. Además, se ha podido comprobar que el mejor individuo encontrado por los EDAs coincidía con el mejor de los obtenidos por el WalkSAT. En las pruebas se ha utilizado el algoritmo UMDA de los EDA sin opciones adicionales, con adaptación de pesos y con optimización local. En el caso de optimización local, como es lógico, aumentaba considerablemente el número de individuos evaluados así como el tiempo de ejecución, pero en ningún caso se ha conseguido mejorar los resultados obtenidos por WalkSAT.

5.8 Conclusiones

El objetivo de este capítulo ha sido analizar diferentes formas de enfocar cómo resolver un problema complejo mediante EDAs y EBCOAs. Para ello se ha elegido el problema de la Satisfactibilidad. Se ha comenzado con una revisión de los algoritmos específicos para resolver este problema con el propósito de conocer las características del problema que más influyen y ayudan en su resolución. En este sentido, se ha encontrado que la mayoría de estos algoritmos ponen especial interés en el orden en que se resuelven cada una de las variables del problema. Se ha comprobado que otros aspectos importantes a tener en cuenta son la actividad de la variable y conectividad de la cláusula.

También se han analizado diferentes tipos de representaciones que habían sido ya utilizadas anteriormente en la resolución del SAT mediante GAs u otros algoritmos evolutivos y se han probado con EDAs y EBCOAs. En este caso se ha podido comprobar que la representación que mejor se adapta a la resolución de este problema mediante computación evolutiva ha sido la representación en la que cada variable del individuo representa a una variable del problema SAT (representación mediante cadena de bits).

Además, se ha probado a mejorar estos resultados mediante técnicas adicionales al proceso evolutivo tales como optimización local y adaptación de la función objetivo mediante pesos. La optimización local ayuda de manera considerable en la resolución del problema ya que profundiza en regiones concretas del espacio de búsqueda. En el caso de la adaptación de la función objetivo, al modificar los pesos de cada cláusula, se produce un efecto similar al backtracking, ya que se puede modificar el orden en que se resuelve cada variable y por tanto, el valor que se le asigna. De esta manera se consigue guiar al algoritmo para salir de óptimos locales.

La idea de utilizar permutaciones para establecer el orden en que se resuelven tanto las cláusulas como las variables, mejora los resultados obtenidos al utilizar representaciones discretas. Esto se puede comprobar comparando los datos de la tabla 5.7 con los correspondientes de las tablas 5.1 y 5.4. Los resultados obtenidos al utilizar permutaciones son mejores a los obtenidos con las representaciones discretas. Sin embargo, de los experimentos realizados podríamos concluir que esto se debe, en gran medida, a la ayuda que proporciona la utilización de información propia del problema a la hora de asignar los valores a las variables. Hay que tener en cuenta, que el proceso de aprendizaje, tanto en EDAs como en EBCOAs, establece las relaciones de interdependencia entre las variables.

Por último se ha planteado la posibilidad de realizar combinaciones de EDAs con algoritmos de búsqueda local. Para ello se han presentado tres formas diferentes de plantear la hibridación. Los resultados, en este caso, llevan a concluir que la búsqueda local por si sola obtiene muy buenos resultados a la hora de afrontar el problema y que estos no es posible mejorarlos mediante EDAs híbridos.

Capítulo 6

Conclusiones y trabajo futuro

En esta tesis se presentan diversos métodos para mejorar el rendimiento de los EDAs, profundizando en el proceso de aprendizaje. La contribución principal consiste en el desarrollo de un conjunto de algoritmos evolutivos (EBCOAs) que combinan técnicas evolutivas y clasificadores Bayesianos para solucionar problemas de optimización. Estos algoritmos se diferencian de los EDAs clásicos en la forma en que se lleva a cabo el proceso de aprendizaje del modelo gráfico probabilístico que representa las relaciones entre las variables. Los EBCOAs clasifican a los diferentes individuos según el valor obtenido por éstos al ser evaluados e incorporan una nueva variable, la clase a la que pertenece cada individuo, al modelo gráfico que se utilizará para instanciar a la nueva población. De esta manera, se consigue añadir al modelo mayor información sobre los diferentes individuos seleccionados para el aprendizaje

Se han desarrollado 5 versiones diferentes de EBCOAs, utilizando en cada caso diferentes clasificadores Bayesianos, para trabajar con problemas que utilizan variables discretas y otras 3 versiones para trabajar con variables continuas. Teniendo en cuenta los resultados experimentales, podemos concluir que en entornos discretos, el clasificador *tree augmented naïve Bayes*, ha dado muy buenos resultados. En el caso de las versiones continuas, el comportamiento de los diferentes clasificadores utilizados por los EBCOAs es diferente dependiendo del tipo de problema que se está resolviendo. En general, tienen un comportamiento comparable al de los EDAs.

También se ha realizado un análisis sobre la influencia de determinados parámetros en el rendimiento de estos nuevos algoritmos. De los resultados de este análisis podemos concluir que parámetros como el tamaño de población o la forma en que se combinan los individuos de una población con los nuevos individuos muestreados para formar la nueva población, influyen de manera importante en el rendimiento de estos algoritmos. Sin embargo, la influencia que ejercen estos parámetros en cada uno de los algoritmos analizados es diferente, por lo que no se pueden sacar conclusiones generales. En este sentido, se ha comprobado que para las funciones continuas analizadas, el clasificador *tree augmented naïve Bayes* obtiene los mejores resultados con tamaños de población considerablemente menores a los que necesita el clasificador *naïve Bayes*

para converger.

Con el propósito de profundizar un poco más en las posibilidades de los EBCOAs y EDAs en general para resolver problemas de optimización complejos se han analizado diferentes formas de afrontar un problema concreto para su resolución mediante EDAs y EBCOAs. Para ello se ha elegido un problema NP-completo, como el problema de la satisfactibilidad (SAT).

Se ha comenzado con una adaptación a los EDAs de diferentes formas de representar el problema, utilizadas ya en la resolución de este problema por otros algoritmos evolutivos como son los algoritmos genéticos. En los experimentos realizados con estas representaciones se ha podido comprobar que la representación mediante secuencia de bits, en la que cada variable del individuo representa una variable del problema a resolver, es la que mejor se adapta al proceso de aprendizaje de los EDAs.

En la resolución de este problema, el algoritmo evolutivo por si solo tiene dificultades en encontrar el óptimo y resulta muy eficiente completarlo mediante otras técnicas adicionales que ayuden al proceso evolutivo. En este sentido, se ha utilizado optimización local y adaptación dinámica de la función objetivo. Mediante la optimización local se consigue profundizar en regiones concretas del espacio de búsqueda. La adaptación de la función objetivo ayuda al algoritmo a salir de óptimos locales. Utilizando optimización local se consigue mejorar los resultados aunque aumentando considerablemente el coste computacional.

Analizando otros algoritmos específicos en la resolución de SAT, se ha observado la importancia que tiene el orden en que se asignan los valores a las variables del problema. En un intento de aprovechar esta característica, se han utilizando nuevas representaciones en las que se busca el orden más apropiado para asignar valores. En estas nuevas representaciones, cada individuo representa una permutación que indica el orden en que se asignarán los valores a las variables. Se han utilizado dos tipos diferentes de representación mediante permutaciones, a las que también se ha añadido optimización local y adaptación de la función objetivo. Los resultados obtenidos en este caso mejoran a los que se habían obtenido con la representación mediante secuencia de bits. Sin embargo, analizando con detalle estos resultados, podemos concluir que esta mejora se debe a que se utiliza información propia del problema para elegir el valor que se le asigna a cada variable y no al proceso de aprendizaje de los EDAs. Este tipo de representaciones no son las más adecuadas para aumentar el rendimiento de los EDAs, ya que no son capaces de encontrar relaciones de dependencia entre las variables del problema y por tanto no sirven de guía en el proceso de aprendizaje.

En cualquier caso, el comportamiento de estos algoritmos evolutivos, al ser algoritmos generales pensados para resolver problemas de optimización en general, no puede compararse al de algoritmos específicos diseñados expresamente para resolver este problema en particular.

Por último, se han realizado hibridaciones de EDAs con otros algoritmos específicos para resolver SAT. Concretamente se han combinado EDAs con dos algoritmos de búsqueda local, que han demostrado obtener buenos resultados en la resolución de SAT,

de tres formas diferentes. El objetivo de esta hibridación ha sido potenciar, mediante ambas técnicas (EDAs y algoritmos específicos de búsqueda local), el rendimiento de cada uno de ellos. En los experimentos realizados, las diferentes hibridaciones de EDAs no han conseguido mejorar el comportamiento de los algoritmos específicos.

Como consecuencia de los resultados obtenidos en los diferentes métodos utilizados para resolver SAT, podemos concluir que este problema, al ser un problema multimodal, ya que presenta diferentes óptimos locales y en algunos casos varios óptimos globales, resulta un problema poco apropiado para resolverlo utilizando algoritmos de estimación de distribuciones debido a que éstos no encuentran diferencias significativas entre individuos que representan regiones muy diferentes dentro del espacio de búsqueda.

Esta tesis también deja líneas futuras de trabajo, tanto en el desarrollo y mejora de los nuevos algoritmos como en nuevas formas de afrontar el problema SAT utilizando algoritmos de estimación de distribuciones.

Con respecto a los algoritmos desarrollados, se plantea probar con nuevos clasificadores Bayesianos, sobre todo en el dominio continuo, capaces de tener en cuenta un mayor número de interdependencias entre variables como pueden ser las multirredes Bayesianas (Bayesian multinets) (Kontkanen y col., 2000).

Por otro lado, se ha podido comprobar que los EBCOAs resultan muy sensibles a ciertos parámetros. Esto deja abierta una posible línea de trabajo sobre los valores más adecuados en cada caso, incluso sobre nuevas formas de combinar poblaciones o de seleccionar las clases.

Desde un punto de vista más general, se podría mejorar el rendimiento de los EBCOAs en entornos continuos, sustituyendo la variable discreta clase, en la red condicional Gaussiana, por el valor obtenido por el individuo al ser evaluado. De esta manera, todos los individuos en la población se pueden agrupar en relación a su verdadero valor. Sin embargo, esta ampliación requiere una cuidadosa consideración debido a los elevados costes computacionales que supone.

En cuanto a la resolución del problema SAT, se podrían trabajar nuevas formas de resolver el problema, como crear una estructura inicial de red, basada en el aprendizaje de las características de problema concreto a resolver, de manera que el algoritmo evolutivo comience con esta estructura en vez de crear, la primera vez, una distribución de probabilidades aleatoria.

Por otro lado, se pueden utilizar EDAs desarrollados específicamente para superar los problemas que presenta la optimización de problemas multimodales (Peña y col., 2005) y comprobar su comportamiento en la resolución de SAT.

Bibliografía

Ackley, D. H. (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluwer.

Armañanzas, R., Inza, I., Santana, R., Saeys, Y., Flores, J. L., Lozano, J. A., Van de Peer, Y., Blanco, R., Robles, V., Bielza, C., y Larrañaga, P. (2008). A review of estimation of distribution algorithms in bioinformatics. *BioData Mining*, 1(6).

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK.

Bäck, T., Eiben, A., y Vink, M. E. (1998). A superior evolutionary algorithm for 3-SAT. En *Proceedings of the 7th Annual Conference on Evolutionary Programming, number 1477 in LNCS*, pp 125–136. Springer.

Ballester, P.J., Stephenson, J., y Gallagher, J.N. Carterand K. (2005). Real-parameter optimization performance study on the CEC-2005 benchmark with SPC-PNX. En *Proceedings of the IEEE International Congress on Evolutionary Computation*, pp 498–505.

Baluja, S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon Report, CMU-CS-94-163.

Baluja, S. y Davies, S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical report, Carnegie Mellon Report, CMU-CS-97-107.

Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C., y Stewart, W.R. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32.

Bengoetxea, E. (2003). *Inexact Graph Matching Using Estimation of Distribution Algorithms*. PhD thesis, Ecole Nationale Supérieure des Télécommunications.

Bengoetxea, E., Larrañaga, P., Bloch, I., y Perchant, A. (2001a). Solving graph matching with EDAs using a permutation-based representation. En Larrañaga, P. y

Lozano, J. A., editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pp 243–265. Kluwer Academic Publishers.

Bengoetxea, E., Miquélez, T., Larrañaga, P., y Lozano, J. A. (2001b). Experimental results in function optimization with EDAs in continuous domain. En Larrañaga, P. y Lozano, J. A., editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pp 181–194. Kluwer Academic Publishers.

Blum, A. L. y Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271.

Bosman, P. A. y Grahl, J. (2008). Matching inductive search bias and problem structure in continuous estimation of distribution algorithms. *European Journal of Operational Research*, 185:1246–1264.

Bosman, P. A. y Thierens, D. (2006). Numerical optimization with real-valued estimation-of-distribution algorithms. En Pelikan, Martin, Sastry, Kumara, y Cantú-Paz, Erick, editors, *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, Studies in Computational Intelligence, pp 91–120. Springer-Verlag.

Bouckaert, R. R. (1994). A stratified simulation scheme for inference in Bayesian belief networks. En *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pp 110–117. Seattle, WA.

Bouckaert, R. R., Castillo, E., y Gutiérrez, J. M. (1996). A modified simulation scheme for inference in Bayesian networks. *International Journal of Approximate Reasoning*, 14:55–80.

Box, G. E. P. y Muller, M. E. (1958). A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29:610–611.

Buntine, W. (1991). Theory refinement in Bayesian networks. En *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pp 52–60.

Cestnik, B., Kononenko, I., y Bratko, I. (1987). ASSISTANT-86: A knowledge elicitation tool for sophisticated users. En Bratko, I. y Lavrac, N., editors, *Progress in Machine Learning*, pp 31–45, Wilmslow, U.K. Sigma Press.

Chow, C. y Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467.

Cook, S. (1971). The complexity of theorem-proving procedures. En Grefenstette, J.J., editor, *In Proceedings of Third Annual ACM Symposium on Theory of Computing*, pp 151–158, New York.

Cooper, G. F. y Herskovits, E. A. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347.

Cotta, C., Alba, E., Sagarna, R., y Larrañaga, P. (2001). Adjusting weights in artificial neural networks using evolutionary algorithms. En Larrañaga, P. y Lozano, J. A., editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pp 361–377. Kluwer Academic Publishers.

Cover, T. M. y Thomas, J. A. (1991). *Elements of Information Theory*. John Wiley and Sons, New York.

Davis, M., Logemann, G., y Loveland, D. (1962). A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397.

Davis, M. y Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215.

Dawid, A. P. (1979). Conditional independence in statistical theory. *Journal of the Royal Statistics Society, Series B*, 41:1–31.

de Bonet, J. S., Isbell, C. L., y Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. En *Advances in Neural Information Processing Systems*, volume 9, pp 424–431, Cambridge, MA. M. Mozer, M. Jordan and Th. Petsche eds.

de Groot, M. (1970). *Optimal Statistical Decisions*. McGraw–Hill, New York.

de Jong, K. y Spears, W. (1989). Using genetic algorithms to solve NP-complete problems. En Schaffer, J.D., editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pp 124–132, San Mateo, CA. Morgan Kaufmann.

Dempster, A. P. (1972). Covariance selection. *Biometrika*, 32:95–108.

Domingos, P. y Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130.

Duda, R. y Hart, P. (1973). *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York.

Durairaj, V. y Kalla, P. (2004). Guiding CNF-SAT search via efficient constraint partitioning. En *International Conference on Computer-Aided Design (ICCAD'04)*.

Durairaj, V. y Kalla, P. (2006). Guiding CNF-SAT search by analyzing constraint-variable dependencies and clause lengths. En *High-Level Design Validation and Test Workshop (HLDVT'06)*.

Echegoyen, C., Lozano, J. A., Santana, R., y Larrañaga, P. (2007). Exact Bayesian network learning in estimation of distribution algorithms. En *Proceedings of the 2007 Congress on Evolutionary Computation CEC-2007*, pp 1051–1058. IEEE Press.

Echegoyen, C., Santana, R., Lozano, J. A., y Larrañaga, P. (2008). The impact of probabilistic learning algorithms in EDAs based on Bayesian networks. En *Linkage in Evolutionary Computation*, Studies in Computational Intelligence, pp 109–139. Springer.

Eiben, A.E. y van der Hauw, J.K. (1997). Solving 3-SAT by GAs adapting constraint weights. En *Proceedings of the IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*.

Etxeberria, R. y Larrañaga, P. (1999). Global optimization with Bayesian networks. En *Special Session on Distributions and Evolutionary Optimization*, pp 332–339, La Habana, Cuba. II Symposium on Artificial Intelligence, CIMAF99.

Fang, L. y Hsiao, M. S. (2007). A new hybrid solution to boost SAT solver performance. En *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pp 1307–1313, San Jose, CA, USA. EDA Consortium.

Ferris, B. y Froehlich, J. (2004). Walksat as an informed heuristic to DPLL in SAT solving. Technical Report CSE 573, Artificial Intelligence.

Fiala, F. (1978). Vehicle routing problems. *GMD-Mitteilungen*, 46.

Flood, M. M. (1956). The traveling salesman problem. *Operations Research*, 4:61–75.

Freeman, J.W. (1995). *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania.

Freisleben, Bernd y Merz, Peter (1996). A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. En *International Conference on Evolutionary Computation*, pp 616–621.

Friedman, N., Geiger, D., y Goldsmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29(2):131–163.

Fung, R. M. y Chang, K. C. (1990). Weighting and integrating evidence for stochastic simulation in Bayesian networks. En Henrion, M., Shachter, R. D., Kanal, L. N., y Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence*, volume 5, pp 209–220, Amsterdam. Elsevier.

Fung, R. M. y del Favero, B. (1994). Backward simulation in Bayesian networks. En *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp 227–234. Morgan Kaufmann Publishers, San Francisco.

Gámez, J. A., Mateo, J. L., y Puerta, J. M. (2007). EDNA: Estimation of dependency networks algorithm. En Mira, J. y Álvarez, J. R., editors, *Bio-inspired Modeling of Cognitive Tasks, Second International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2007*, volume 4527 of *Lecture Notes in Computer Science*, pp 427–436. Springer Verlag.

Gammerman, A. y Thatcher, A. R. (1991). Bayesian diagnostic probabilities without assuming independence of symptoms. *Methods of Information in Medicine*, 30(1):15–22.

Ganai, M. K., Zhang, L., Ashar, P., Gupta, A., y Malik, S. (2002). Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. *Design Automation Conference*, 0:747.

Garey, M.R. y Johnson, D.S. (1979). *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Geiger, D. y Heckerman, D. (1994). Learning Gaussian networks. Technical report, Microsoft Advanced Technology Division, Microsoft Corporation, Seattle, Washington.

Giunchiglia, E., Giunchiglia, F., y Tacchella, A. (2000). SAT-based decision procedures for classical modal logics. En Gent, I.P., van Maaren, H., y Walsh, T., editors, *SAT 2000, Highlights of Satisfiability Research in the Year 2000, Frontiers in Artificial Intelligence and Applications*. IOS Press.

Goldberg, E. y Novikov, Y. (2002). Berkmin: A fast and robust sat solver. En *Proceedings of Design, Automation and Test in Europe (DATE'02)*, pp 142–149.

Gottlieb, J., Marchiori, E., y Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50.

Gottlieb, J. y Voss, N. (1998a). Improving the performance of evolutionary algorithms for the satisfiability problem by refining functions. En *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 1498 of *Lecture Notes in Computer Science*, pp 755–764, Berlin, Germany. Springer.

Gottlieb, J. y Voss, N. (1998b). Representations, fitness functions and genetic operators for the satisfiability problem. En *Proceedings of Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pp 55–68, Berlin, Germany. Springer.

Gottlieb, J. y Voss, N. (2000). Adaptive fitness functions for the satisfiability problem. En *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature (PPSN'00)*, volume 1917 of *Lecture Notes in Computer Science*, pp 621–630, Berlin, Germany. Springer.

Gu, J. y Puri, R. (1995). Asynchronous circuit synthesis with boolean satisfiability. *IEEE Transactions on Evolutionary Computation*, 14(8):961–973.

Hansen, N. y Kern, S. (2004). Evaluating the CMA evolution strategy on multimodal test functions. En *Eighth International Conference on Parallel Problem Solving from Nature – PPSN VIII*, pp 282–291.

Hansen, N., Müller, S. D., y Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18.

Hao, J.-K. (1995). A clausal genetic representation and its evolutionary procedures for satisfiability problems. En Pearson, D. W., Steel, N. C., y Albrecht, R. F., editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pp 289–292, Vienna, Austria. Springer.

Hao, J.-K., Lardeux, F., y Saubion, F. (2003). Evolutionary computing for the satisfiability problem. En *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pp 259–268.

Harik, G., Lobo, F. G., y Golberg, D. E. (1998). The compact genetic algorithm. En *Proceedings of the IEEE Conference on Evolutionary Computation*, pp 523–528, Piscataway, NJ.

Hauschild, M., Pelikan, M., Lima, C., y Sastry, K. (2007). Analyzing probabilistic models in hierarchical BOA on traps and spin glasses. En *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume I, pp 523–530.

Heckerman, D., Geiger, D., y Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243.

Heckerman, D. y Wellman, M. P. (1995). Bayesian networks. *Communications of the ACM*, 38:27–30.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. *Uncertainty in Artificial Intelligence*, 2:149–163. J. F. Lemmer and L. N. Kanal eds., North-Holland, Amsterdam.

Herves, V., Larrañaga, P., Robles, V., Peña, J. M., Pérez, M. S., y Rosales, F. (2004). EDA paralelos multipoblación para el problema SAT. En *Proceedings of the III Spanish Conference on Metaheuristics and Evolutive and Bioinspired Algorithms*, Cordoba (Spain). (in Spanish).

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Michigan.

Hoos, H. y Stützle, T. (2000). SATLIB: An online resource for research on SAT. En *SAT 2000*, pp 283–292. IOS Press.

Howard, R. y Matheson, J. (1981). Influence diagrams. En Howard, R. y Matheson, J., editors, *Readings on the Principles and Applications of Decision Analysis*, volume 2, pp 721–764. Strategic Decision Group, Menlo Park, California.

Inza, I., Larrañaga, P., Etxeberria, R., y Sierra, B. (2000). Feature subset selection by Bayesian networks based optimization. *Artificial Intelligence*, 123(1-2):157–184.

Kaufman, K. y Michalski, R. (1999). The AQ18 machine learning and data mining system: An implementation and user's guide. Technical report, Machine Learning and Inference Laboratory, George Manson University, Fairfax, Virginia.

Kautz, H., Sabharwal, A., y Selman, B. (2009). Incomplete algorithms. En Biere, A., Heule, M., van Maaren, H., y Walsh, T., editors, *Handbook of Satisfiability*, volume 185, pp 185–203. IOS Press.

Kautz, H. y Selman, B. (1992). Planning as satisfiability. En *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pp 359–363.

Kohavi, R. (1995). *Wrappers for Performance Enhancements and Oblivious Decision Graphs.* PhD thesis, Stanford University, Stanford, CA, USA.

Kohavi, R. y John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.

Kononenko, I. (1990). Comparison of inductive and naïve Bayesian learning approaches to automatic knowledge acquisition. En Wielinga, B., Boose, J., Gaines, B., Shereiber, G., y van Someren, M., editors, *Current Trends in Knowledge Acquisition*, pp 190–197, Amsterdam. IOS Press.

Kononenko, I. (1991). Semi-naïve Bayesian classifiers. En *Proceedings of the 6th European Working Session on Learning*, pp 206–219, Porto, Portugal.

Kontkanen, P., Myllymäki, P., Tirri, H., y Valtonen, K. (2000). Bayesian multinet classifiers. En *Proceedings of the 10th International Conference on Computing and Information – ICCI 2000.*

Kruskal, W.H. y Wallis, W.A. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621.

Kuehlmann, A., Ganai, M. K., y Paruthi, V. (2001). Circuit-based boolean reasoning. En *DAC '01: Proceedings of the 38th annual Design Automation Conference*, pp 232–237, New York, NY, USA. ACM.

Langley, P. y Sage, S. (1994). Induction of selective Bayesian classifiers. En *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pp 399–406, Seattle, WA.

Lardeux, F., F.Saubion, y Hao, J.-K. (2006). Gasat: a genetic local search algorithm for the satisfiability problem. *Evolutionary Computation*, 14(2):223–253.

Larrañaga, P., Etxeberria, R., Lozano, J. A., y Peña, J. M. (2000). Optimization in continuous domains by learning and simulation of Gaussian networks. En *Proceedings of the Workshop in Optimization by Building and using Probabilistic Models. A Workshop within the 2000 Genetic and Evolutionary Computation Conference, GECCO 2000*, pp 201–204, Las Vegas, Nevada, USA.

Larrañaga, P. y Lozano, J. A. (2001). *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

Larrañaga, P., Lozano, J. A., y Bengoetxea, E. (2001). Estimation of distribution algorithms based on multivariate normal and Gaussian networks. Technical Report KZZA-IK-1-01, Department of Computer Science and Artificial Intelligence, University of the Basque Country.

Lauritzen, S. L. (1996). *Graphical Models*. Oxford University Press.

Letombe, F. y Marques-Silva, J. (2008). Improvements to hybrid incremental SAT algorithms. En *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT´08)*, volume 4996 of *Lecture Notes in Computer Science*, pp 168–181, Berlin / Heidelberg. Springer.

Lewis, D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval. En Nédellec, Claire y Rouveirol, Céline, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pp 4–15, Chemnitz, DE. Springer Verlag, Heidelberg, DE.

Li, C.M. y Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. En *Proceedings of IJCAI 97*, pp 366–371.

Lima, C. F., Pelikan, M., Goldberg, D. E., Lobo, F. G., Sastry, K., y Hauschild, M. (2007). Influence of selection and replacement strategies on linkage learning in BOA. En *Proceedings of the 2007 Congress on Evolutionary Computation CEC-2007*, pp 1083–1090. IEEE Press.

Liu, H. y Motoda, H. (1998). *Feature Selection. From Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Boston.

Llorà, X. y Goldberg, D.E. (2003). Wise breeding GA via machine learning techniques for function optimization. En et al., Cantú-Paz, editor, *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-03, Part I*, Lecture Notes in Computer Science 2723, pp 1172–1183, Chicago, Illinois. Springer.

Lozano, J. A., Sagarna, R., y Larrañaga, P. (2001). Parallel estimation of distribution algorithms. En Larrañaga, P. y Lozano, J. A., editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pp 129–145. Kluwer Academic Publishers.

Lozano, J.A., Larrañaga, P., Inza, I., y Bengoetxea, E. (2006). *Towards a New Evolutionary Computation. Advances in Estimation of Distribution Algorithms*. Springer.

Marchiori, E. y Rossi, C. (1999). A flipping genetic algorithm for hard 3-SAT problems. En et al., W. Banzhaf, editor, *Proceedings of Genetic and Evolutionary Computation Conference*, pp 393–400, San Francisco, CA. Morgan Kaufmann.

Marques-Silva, J. y Sakallah, K.A. (1996). GRASP - a new search algorithm for satisfiability. En *International Conference on Computer-Aided Design (ICCAD'96)*, pp 220–227.

Marques-Silva, J. y Sakallah, K.A. (1999). GRASP - a new search algorithm for satisfiability. *IEEE Transactions on Computers*, 48(5):506–521.

McAllester, D., Selman, B., y Kautz, H. (1997). Evidence for invariants in local search. En *Proceedings of the 14th National (US) Conference on Artificial Intelligence (AAAI'97)*, pp 321–326. The MIT Press.

Mendiburu, A. (2006). *Parallel Implementation of Estimation of Distribution Algorithms based on Probabilistic Graphical Models. Application to Chemical Calibration Models*. PhD thesis, University of the Basque Country.

Mendiburu, A., Santana, R., Bengoetxea, E., y Lozano, J. (2007). A parallel framework for loopy belief propagation. En *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2007*, volume II, pp 2843–2850.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, Berlin Heidelberg.

Michalski, R.S. (2000). Learnable evolution model: Evolutionary processes guided by machine learning. *Machine Learning*, 38:9–40.

Minsky, M. (1961). Steps toward artificial intelligence. *Transactions on Institute of Radio Engineers*, 49:8–30.

Miquélez, T., Bengoetxea, E., y Larrañaga, P. (2004). Evolutionary computation based on Bayesian classifiers. *International Journal of Applied Mathematics and Computer Science*, 14(3):335–349.

Miquélez, T., Bengoetxea, E., Mendiburu, A., y Larrañaga, P. (2007). Combining Bayesian classifiers and estimation of distribution algorithms for optimization in continuous domains. *Connection Science*, 19(4):297–319.

Mitchell, D. G., Selman, B., y Levesque, H. J. (1992). Hard and easy distributions for SAT problems. En Rosenbloom, P. y Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp 459–465, Menlo Park, California. AAAI Press.

Moskewicz, M., Madigan, C., Zhao, L., y Malik, S. (2001). CHAFF: Engineering and efficient SAT solver. En *Proceedings of the 38th Design Automation Conference (DAC'01)*, pp 530–535.

Mühlenbein, H. (1998). The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346.

Mühlenbein, H. y Höns, R. (2005). The estimation of distributions and the minimum relative entropy principle. *Evolutionary Computation*, 13(1):1–27.

Mühlenbein, H. y Mahning, T. (1999). FDA - a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376.

Mühlenbein, H., Mahning, T., y Ochoa, A. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):215–247.

Mühlenbein, H. y Paaß, G. (1996). From recombination of genes to the estimation of distributions: I. Binary parameters. En et al., M. Voigt, editor, *Parallel Problem Solving from Nature - PPSN IV. Lecture Notes in Computer Science 1411*, pp 178–187.

Muñoz, A. (2003). LEM algorithms. MSc. thesis, Computer Engineering Faculty, University of the Basque Country.

Neapolitan, R.E. (2003). *Learning Bayesian Networks*. Prentice Hall.

Nilsson, D. (1998). An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 2:159–173.

Ohmann, C., Yang, Q., Kunneke, M., Stolzing, H., Thon, K., y Lorenz, W. (1988). Bayes theorem and conditional dependence of symptoms: different models applied to data of upper gastrointestinal bleeding. *Methods of Information in Medicine*, 27(2):73–83.

Pazzani, M. (1997). Searching for dependencies in Bayesian classifiers. En Fisher, D. y Lenz, H.-J., editors, *Learning from Data: Artificial Intelligence and Statistics V*, pp 239–248, New York, NY. Springer–Verlag.

Pearl, J. (1985). Bayesian networks: A model of self-activated memory for evidential reasoning. Technical Report CSD-850021, R-43, UCLA Computer Science Department.

Pearl, J. (1987). Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32(2):245–257.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California.

Pelikan, M. y Goldberg, D. E. (2003). Hierarchical BOA solves ising spin glasses and MAXSAT. En *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), number 2724 in LNCS*, pp 1271–1282. Springer.

Pelikan, M., Goldberg, D. E., y Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm. En *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Orlando FL*, volume 1, pp 525–532, San Francisco, California. Morgan Kaufmann Publishers.

Pelikan, M., Goldberg, D. E., y Cantú-Paz, E. (2000). Bayesian optimization algorithm, population sizing, and time to convergence. En *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2000*, pp 275–282.

Pelikan, M. y Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. En Roy, R., Furuhashi, T., y Chandhory, P. K., editors, *Advances in Soft Computing-Engineering Design and Manufacturing*, pp 521–535, London. Springer-Verlag.

Pelikan, M., Sastry, K., y Cantú-Paz, E., editors (2006). *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*. Studies in Computational Intelligence. Springer, Berlin.

Peña, J. M., Lozano, J. A., y Larrañaga, P. (2005). Globally multimodal problem optimization via an estimation of distribution algorithm based on unsupervised learning of bayesian networks. *Evolutionary Computation*, 13(1):43–66.

Pérez, A., Larrañaga, P., y Inza, I. (2006). Supervised classification with conditional Gaussian networks: Increasing the structure complexity from naive Bayes. *International Journal of Approximate Reasoning*, 43:1–25.

Pipatsrisawat, K. y Darwiche, A. (2007). RSAT 2.0: SAT solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA.

Rana, Soraya y Whitley, Darrell (1998). Genetic algorithm behavior in the MAXSAT domain. *Lecture Notes in Computer Science*, 1498:785–794.

Ripley, B. D. (1987). *Stochastic Simulation*. John Wiley and Sons.

Robles, V., de Miguel, P., y Larrañaga, P. (2001). Solving the travelling salesman problem with estimation of distribution algorithms. En Larrañaga, P. y Lozano, J. A., editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, pp 211–229. Kluwer Academic Publishers.

Romero, T. y Larrañaga, P. (2009). Triangulation of Bayesian networks with recursive estimation of distribution algorithms. *International Journal of Approximate Reasoning*, 50(3):472–484.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3:175–184.

Rossi, C., Marchiori, E., y Kok, J. (2000). An adaptive evolutionary algorithm for the satisfiability problem. En *Proceedings of the ACM symposium on Applied computing (SAC '00)*, pp 463–469, New York, NY, USA. ACM Press.

Rudlof, S. y Köppen, M. (1996). Stochastic hill climbing by vectors of normal distributions. En *Proceedings of the First Online Workshop on Soft Computing (WSC1)*, Nagoya, Japan.

Sagarna, R. y Lozano, J. A. (2006). Scatter search in software testing, comparison and collaboration with estimation of distribution algorithms. *European Journal of Operational Research*, 169:392–412.

Sahami, M. (1996). Learning limited dependence Bayesian classifiers. En *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp 335–338, Portland, OR.

Salomon, R. (1998). Evolutionary algorithms and gradient search: similarities and differences. *IEEE Transactions on Evolutionary Computation*, 2(2):45–55.

Santana, R. (2003). Factorized distribution algorithms: selection without selected population. Technical Report ICIMAF 2003-240, Institute of Cybernetics, Mathematics and Physics, Havana, Cuba.

Santana, R. (2005). Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, 13(1):67–97.

Santana, R., Larrañaga, P., y Lozano, J. A. (2005). Interactions and dependencies in estimation of distribution algorithms. En *Proceedings of the 2005 Congress on Evolutionary Computation CEC-2005*, pp 1418–1425, Edinburgh, U.K. IEEE Press.

Santana, R., Larrañaga, P., y Lozano, J. A. (2006). Mixtures of Kikuchi approximations. En Fürnkranz, Johannes, Scheffer, Tobias, y Spiliopoulou, Myra, editors, *Proceedings of the 17th European Conference on Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Artificial Intelligence*, pp 365–376. Springer Verlag.

Santana, R., Larrañaga, P., y Lozano, J. A. (2008). Adaptive estimation of distribution algorithms. En Cotta, C., Sevaux, M., y Sörensen, K., editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pp 177–197. Springer-Verlag.

Santana, R., Larrañaga, P., y Lozano, J. A. (2009). Research topics in discrete estimation of distribution algorithms based on factorizations. *Memetic Computing*, 1(1):35–54.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 7(2):461–464.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. Wiley InterScience.

Sebag, M. y Ducoulombier, A. (1998). Extending population-based incremental learning to continuous search spaces. En Bäck, T., Eiben, G., Schoenauer, M., y Schwefel, H.-P., editors, *Proceedings of the 5th Conference on Parallel Problem Solving from Nature – PPSN V*, pp 418–427, Berlin. Springer-Verlag.

Selman, B., Kautz, H. A., y Cohen, B. (1996). Local search strategies for satisfiability testing. En *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp 521–532.

Selman, B., Kautz, H.A., y Cohen, B. (1994). Noise strategies for improving local search. En *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pp 337–343, Seattle.

Selman, B., Levesque, H.J., y Mitchell, D. (1992). A new method for solving hard satisfiability problems. En Rosenbloom, P. y Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp 440–446, Menlo Park, California. AAAI Press.

Shachter, R. y Kenley, C. (1989). Gaussian influence diagrams. *Management Science*, 35(5):527–550.

Shachter, R.D. y Peot, M.A. (1990). Simulation approaches to general probabilistic inference on belief networks. En *Uncertainty in Artificial Intelligence 5*, pp 221–234. Elsevier, Amsterdam.

Shwe, M. y Cooper, G. (1991). An empirical analysis of likelihood-weighting simulation on a large multiply connected medical belief network. *Computers and Biomedical Research*, 24:453–475.

Soto, M. R. (2003). *A Single Connected Factorized Distribution Algorithm and its Cost of Evaluation*. PhD thesis, University of Havana, Havana, Cuba. In Spanish.

Speed, T. P. y Kiiveri, H. (1986). Gaussian Markov distributions over finite graphs. *Annals of Statistics*, 14:138–150.

Spirtes, P., Glymour, C., y Scheines, R. (1991). An algorithm for fast recovery of sparse causal graphs. *Social Science Computing Reviews*, 9:62–72.

Sörensson, N. y Een, N. (2005). Minisat: A SAT solver with conflict-clause minimization. En *The International Conference on Theory and Applications of Satisfiability Testing*.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36(1):111–147.

Syswerda, G. (1993). Simulated crossover in genetic algorithms. En Whitley, L.D., editor, *Foundations of Genetic Algorithms*, volume 2, pp 239–255, San Mateo, California. Morgan Kaufmann.

Todd, B. S. y Stamper, R. (1994). The relative accuracy of a variety of medical diagnostic programs. *Methods of Information in Medicine*, 33:402–416.

Törn, A. y Zilinskas, A. (1989). *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer.

Ventura, S., Herrera, F., Berná, J.N., y Hervás, C. (2002). Evolución guiada mediante aprendizaje. Comparación en problemas de optimización. En *Proceedings of the Conference on 'Algoritmos Evolutivos y Bioinspirados' (AEB'02)*, pp 430–436.

Wang, X., Wang, H., y Ma, G. (2008). Hybrid SAT solver considering circuit observability. *Young Computer Scientists, International Conference for*, 0:65–70.

Wermuth, N. (1976). Model search among multiplicative models. *Biometrics*, 32:253–263.

Whitley, D. y Kauth, J. (1988). GENITOR: A different genetic algorithm. En *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, volume 2, pp 118–130, Denver, Colorado.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics*. John Wiley and Sons.

Yanover, C. y Weiss, Y. (2004). Finding the M most probable configurations using loopy belief propagation. En Thrun, S., Saul, L., y Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

Yu, T.-L., Sastry, K., Goldberg, D. E., y Pelikan, M. (2007). Population sizing for entropy-based model building in genetic algorithms. En *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2007*, volume I, pp 601–608.

Zhang, H. (1997). SATO: An efficient propositional prover. En *Proceedings of the International Conference on Automated Deduction*, pp 272–275.

Zhang, Q., Zhou, A., y Jin, Y. (2008). Modelling the regularity in estimation of distribution algorithm for continuous multi-objective evolutionary optimization with variable linkages. *IEEE Transactions on Evolutionary Computation*, 12(1):49–63.

Índice alfabético