INSTITUTO DE CIBERNÉTICA, MATEMÁTICA Y FÍSICA
DEPARTAMENTO DE MATEMÁTICA INTERDISCIPLINARIA

HACIA UNA GENERACIÓN MÁS EFICIENTE DE ALGORITMOS EVOLUTIVOS CON ESTIMACIÓN DE DISTRIBUCIONES: PRUEBAS DE INDEPENDENCIA + PARALELISMO

Tesis presentada en opción al grado científico de
Doctor en Ciencias Matemáticas

JULIO CESAR MADERA QUINTA

La Habana
2009

INSTITUTO DE CIBERNÉTICA, MATEMÁTICA Y FÍSICA
DEPARTAMENTO DE MATEMÁTICA INTERDISCIPLINARIA

HACIA UNA GENERACIÓN MÁS EFICIENTE DE ALGORITMOS EVOLUTIVOS CON ESTIMACIÓN DE DISTRIBUCIONES: PRUEBAS DE INDEPENDENCIA + PARALELISMO

Tesis presentada en opción al grado científico de
Doctor en Ciencias Matemáticas

Autor: MSc. Julio Cesar Madera Quintana

Tutores: Dr. Alberto Ochoa Rodríguez
Dr. Enrique Alba Torres

La Habana
2009

DEDICATORIA

A mis padres, esposa e hija

AGRADECIMIENTOS

Agradezco en primer lugar a mis padres, mi hermana, mi hija, mi suegra y al resto de la familia más cercana, sin los cuales esta tesis no hubiera sido posible.

A mi esposa por su apoyo incondicional, sacrificio y ayuda durante todo el tiempo de investigación.

A mis tutores, Dr. Alberto Ochoa Rodríguez y Dr. Enrique Alba Torres, por el tiempo que me han dedicado, por sus consejos, por darme la oportunidad de realizar esta tesis, por su participación en mi formación científica y más que nada por ser mis amigos.

A mis amigos Deisbel, Rigre y Sandro, por estar siempre preocupados por la culminación de esta investigación.

A Geysel, Yumilka y Olguita por su constante preocupación, apoyo y estimulación a la finalización de la investigación.

A Yailé por su apoyo y sacrificio al frente del departamento en mi sustitución.

A Martica por su ayuda en la revisión y conclusión de la escritura de la tesis.

A mis compañeros de trabajo que sin su apoyo y sacrificio no hubiese sido posible este resultado.

A Bernabé, Paco, Gabriel y Francis por la colaboración en muchas de las investigaciones que hemos realizado de forma conjunta.

A todas aquellas personas que permitieron que esta investigación fuera posible.

SÍNTESIS

En esta tesis se investiga una clase de algoritmos evolutivos (EA), los Algoritmos de Estimación de Distribuciones (EDA) en problemas de optimización discreta y continua. La intención principal es el estudio de la eficiencia del EDA desde dos perspectivas: disminuir el número de evaluaciones de la función objetivo (eficiencia evaluativa) y el tiempo de ejecución del algoritmo (eficiencia en tiempo).

Primeramente se reporta un estudio sobre la utilización de los modelos Bayesianos que utilizan detección de independencia basados en restricciones para el aprendizaje de la estructura del modelo probabilístico. Como resultado se crean dos nuevos algoritmos basados en restricciones: EDA que utiliza *maximización-minimización con escalador de colinas* ($\text{CBEDA}_{\text{MMHC}}$) y EDA basado en *detección de dependencias de tres fases* ($\text{CBEDA}_{\text{TPDA}}$). El algoritmo de aprendizaje de $\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{TPDA}}$ primeramente construye un *esqueleto* no orientado haciendo pruebas de independencia y después lo orienta con un proceso de optimización de métrica.

Una vez que se dispone de algoritmos más eficientes en cuanto al número de evaluaciones, el estudio se centra en la reducción del tiempo de ejecución con la creación de algoritmos paralelos. Se crean los algoritmos paralelos $p\text{CBEDA}_{\text{MMHC}}$, $\text{pCBEDA}_{\text{TPDA}}$ y $\text{pCBEDA}_{\text{LPA}}$ bajo un esquema maestro/trabajador y se proponen dos modelos descentralizados: distribuido y celular.

Experimentalmente se muestra que los algoritmos $\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{TPDA}}$ realizan menor número de evaluaciones que otros EDA del estado de arte en el conjunto de funciones de prueba utilizado. Se muestra que la paralelización permite reducir el tiempo de ejecución de los algoritmos; en el caso de las variantes descentralizadas también conlleva a una disminución del número de evaluaciones.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

Lista de algoritmos

LISTA DE ABREVIATURAS

AIC: Aikaike Information Criterion

ANN: Artificial Neural Network

API: Application Programming Interface

BDe: Bayesian-Dirichlet Metric

BIC: Bayesian Information Criterion

BMDA: Bivariate Marginal Distribution Algorithm

BOA: Bayesian Optimization Algorithm

CBEDA: Constraint Based Estimation of Distribution Algorithm

cEA: cellular Evolutionary Algorithm

cEDA: cellular EDA

cGA: compact Genetic Algorithm

CMMHC: Continuous MMHC

COMIT: Combining Optimizers with Mutual Information Trees

COW: Cluster Of Workstations

CPT: Conditional Probability Table

cUMDA: cellular UMDA

DAG: Direct Acyclic Graph

dEDA: distributed EDA

dUMDA: distributed UMDA

dUMDAC: distributed UMDA for Continuous domain

dUMDAD: distributed UMDA for Discrete domain

EA: Evolutionary Algorithms

EBNA: Estimation of Bayesian Network Algorithm

ECGA: Extended Compact Genetic Algorithm

EDNA: Estimation of Dependency Networks Algorithm

EGNA: Estimation of Gaussian Network Algorithm

EMNA: Estimation of Multivariate Normal Algorithm

EP: Evolutionary Programming

ES: Evolutionary Strategies

FDA-learning: Factorized Distribution Algorithm with Learning

FDA: Factorized Distribution Algorithm

GA: Genetic Algorithm

GM: Graphical Models

GP: Genetic Programming

hBOA: hierarchical Bayesian Optimization Algorithm

LCEA: Low Cost Evolutionary Algorithm

LFDA: Learning Factorized Distribution Algorithm

LPA: Learning Polytree Algorithm

MIMIC: Mutual Information Maximization for Input Clustering

MLP: Multi-Layer Perceptron

MMHCEDA: Max-Min Hill-Climbing EDA

MMPC: Max-Min Parents and Childrens

MN-EDA: Markov Network Estimation of Distribution Algorithm

MN-FDA: Markov Network Factorized Distribution Algorithm

MPI: Message Passing Interface

MT-FDA: Mixture Tree Factorized Distribution Algorithm

MWST: Maximum Weigth Spanning Tree

PADA: Polytree Approximation of Distribution Algorithm

PBIL: Population Based Incremental Learning

pBOA: parallel BOA

PGM: Probabilistic Graphical Models

PLS: Probabilistic Logic Sampling

PSP: Protein Structure Prediction

RBUF: Random Univariate B-Functions

Speed-up: Ganancia en velocidad, incremento en velocidad

TA: Approximation Tree Algorithm

UMDA: Univariate Marginal Distribution Algorithm

UMDAC: UMDA for Continuous domain

UMDAD: UMDA for Discrete domain

INTRODUCCIÓN

Los Algoritmos Evolutivos (EA) engloban una familia de técnicas computacionales y comparten una característica común: se inspiran en la evolución natural de las especies. Los EA utilizan una estructura compuesta por un conjunto de variables (genes) que pueden tomar diferentes valores y se le denomina individuo. Cada uno codifica una solución con un valor de adecuación o aptitud (fitness) que lo diferencia de otros. El número de variables y el rango de valores que toman son dependientes del problema. El objetivo del EA es encontrar una solución lo suficientemente cercana al óptimo global.

Dentro de los EA clasifican una serie de métodos: los Algoritmos Genéticos (GA), muy utilizados en la optimización de problemas combinatorios y sustentados en la recombinación (cruzamiento y mutación) [50, 65]; las Estrategias Evolutivas (ES), para la optimización de funciones continuas con recombinación [134]; la Programación Genética (GP), basada en la evolución de programas [67]; la Programación Evolutiva (EP), en la optimización continua sin recombinación [41] y los Algoritmos de Estimación de Distribuciones (EDA) para la optimización de funciones discretas y continuas mediante el aprendizaje de la estructura del problema [104, 71].

Los EDA se basan en el aprendizaje y simulación de distribuciones de probabilidad (en lo adelante, distribuciones), a partir de una población de individuos. La meta es extraer información sobre las posibles relaciones entre las variables del problema a optimizar.

De acuerdo a este esquema general, varios EDA se han propuesto en la última década [128, 39, 76, 102, 125]. Estos algoritmos resuelven exitosamente un rango considerable de problemas [71, 125, 80]. Sin embargo, la ejecución de un EDA puede tomar una cantidad considerable de tiempo. Existen dos razones fundamentales: (1) Los EDA que generalmente obtienen mejores resultados son los que consideran múltiples dependencias entre las variables (utilizan modelos probabilísticos complejos) y (2) la utilización de poblaciones implica la evaluación de cada individuo en la función de aptitud. Esta evaluación toma una cantidad de tiempo importante cuando la función es costosa.

La complejidad de las distribuciones está relacionada con la capacidad para representar las relaciones de independencia entre las variables del problema. Mientras mayor es la complejidad de la distribución que se desea estimar, mayor es el número de puntos que se requiere para su correcta estimación.

Reducir el costo del proceso de búsqueda es una cuestión crítica en EDA. Usualmente este costo se relaciona con el número de evaluaciones de la función objetivo (*costo evaluativo*) y el tiempo de ejecución del algoritmo (*costo en tiempo*).

Una de las líneas de investigación que se desarrollan en el grupo del autor, desde hace varios años, es la exploración de estrategias para crear EDA más eficientes. Dentro de las principales se encuentran: la evaluación parcial [150], mutación entrópica [150, 118], utilización de los modelos simplemente

conectados construidos con algoritmos de aprendizaje basados en pruebas de independencia [151, 152, 115, 150] y el elitismo probabilístico [150]. Siguiendo esta línea de investigación, una motivación es explorar el paralelismo y la distribución como vías para la creación de EDA más eficientes.

Otra motivación y antecedente importante fue el algoritmo PADA [151, 152, 150], desarrollado en el grupo de investigación del autor. El mismo se basa en la detección de independencias para estimar la distribución del conjunto seleccionado. A partir de los buenos resultados mostrados por este algoritmo una motivación fue crear versiones paralelas de PADA para reducir su tiempo de ejecución. Continuando con esta línea, la paralelización se centró en algoritmos que sus versiones secuenciales fueran eficientes. En este sentido ya se contaba con la experiencia de PADA por lo que la investigación se dirigió hacia la paralelización de EDA que utilizan pruebas de independencias para aprender la distribución de búsqueda.

Otros antecedentes y motivaciones fueron los trabajos desarrollados por el grupo de Inteligencia Artificial del País Vasco (España) [81, 82, 96, 98] y la presentación de la tesis doctoral de Ocenasek [110]. Todos estos trabajos buscan dar un salto cualitativo en el campo de los EDA, de los estudios teóricos a las aplicaciones prácticas, muy relacionado con la utilización del paralelismo.

En el campo de los EDA existen varios trabajos teóricos que hacen un estudio de las características de diferentes algoritmos, fundamentalmente de los modelos más simples [104, 101, 150]. No obstante, existen algunas cuestiones que meritan ser estudiadas; en particular, por qué la mayoría de los EDA con mejores resultados numéricos se centran en el aprendizaje estructural mediante optimización de métricas (por ejemplo; BOA [128], hBOA [125], EBNA [39, 76] y LFDA [102]). Producto a esto se desprende que las principales implementaciones paralelas de EDA se basan en optimización de métricas [111, 112, 110, 97, 95].

Estas motivaciones y antecedentes dan origen al tema de esta investigación que se centra en la búsqueda de EDA más eficientes utilizando el paralelismo y la distribución. Este involucra diversos conceptos: eficiencia, algoritmo distribuido, algoritmo paralelo y problema complejo.

El concepto de **eficiencia**, el cual puede ser definido en base a varios parámetros, en este trabajo se utiliza relacionándolo con la disminución del número de evaluaciones (*eficiencia evaluativa*) y con la reducción del tiempo de ejecución (*eficiencia en tiempo*).

Un **algoritmo distribuido** para una colección de P procesos, según Gerard Tel [161], es una colección de algoritmos locales, uno por cada proceso en P. En un trabajo anterior [160], Tel lo definió como: “un algoritmo distribuido se ejecuta como una colección de procesos secuenciales, todos ejecutando su parte del algoritmo de forma independiente pero coordinando su actividad a través de la comunicación". Nancy Lynch argumenta: “Los algoritmos distribuidos están diseñados para ejecutarse en un hardware formado por varios procesadores interconectados. El algoritmo funciona correctamente aún cuando los procesadores individuales y los canales de comunicación operan a diferentes velocidades, incluso cuando alguna de las componentes falla" [83]. En esta tesis se tienen en cuenta estas definiciones de algoritmo distribuido.

En contraste con la definición de algoritmo distribuido está la de **algoritmo paralelo** que en oposición a los algoritmos clásicos o algoritmos secuenciales, es un algoritmo que se ejecuta por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado correcto [68].

Por último, el concepto de **problema complejo** se relaciona con los problemas que tienen: un amplio número de subsoluciones que deben ser descubiertas completamente, subsoluciones que no escalan adecuadamente, presentar varios óptimos locales, un alto grado de interacción entre las subsoluciones y un alto ruido externo o estocasticidad [52].

A partir de la problemática de reducir el costo del EDA se hace necesario encontrar nuevos algoritmos que sean del estado del arte y que disminuyan el número de evaluaciones y el tiempo de ejecución. Varios autores [150, 125, 95, 142, 118] han explorado diferentes estrategias para hacer de los EDA, algoritmos más eficientes. Estos incluyen: la programación paralela, técnicas híbridas, nuevos operadores como el elitismo y la mutación, entre otros.

Desde esta óptica el **problema de investigación** que se asumió puede plantearse de la siguiente manera:

¿Cómo construir algoritmos EDA mediante las tecnologías paralelas y distribuidas, de manera tal que estos sean eficientes desde el punto de vista evaluativo y en tiempo de ejecución?

El **objeto de estudio** son los algoritmos de estimación de distribuciones.

Para dar respuesta a la problemática planteada se propone como **objetivo general**:

Crear esquemas paralelos de EDA con aprendizaje basado en pruebas de independencia y esquemas descentralizados para reducir el número de evaluaciones y el tiempo de ejecución.

Para lograr este objetivo se plantean los siguientes **objetivos específicos**:

1. Crear dos nuevos algoritmos EDA que utilicen pruebas de independencia en el aprendizaje y que permitan disminuir el número de evaluaciones de los algoritmos del estado del arte que utilizan optimización de métricas.
2. Crear dos nuevos algoritmos EDA paralelos basados en pruebas de independencia y en el modelo maestro/trabajador de la programación paralela, para disminuir el tiempo de ejecución de los propuestas secuenciales planteadas en el punto anterior. Crear una versión paralela del algoritmo PADA.
3. Crear dos modelos de EDA descentralizados, el de islas y el celular, que sean más eficientes evaluativamente que el centralizado, y en el caso del distribuido que también disminuya el tiempo de ejecución.

El **campo de acción** está relacionado con los algoritmos EDA paralelos basados en pruebas de independencia y los EDA descentralizados.

La **hipótesis** del trabajo es la siguiente:

Si se diseñan y desarrollan algoritmos EDA que utilizan las tecnologías paralelas y distribuidas para implementar esquemas paralelos basados en la detección de independencias y esquemas

descentralizados. Los primeros aumentan la eficiencia en tiempo mientras que los segundos también pueden aumentar la eficiencia evaluativa.

Las **tareas de investigación** trazadas son:

1. Caracterizar los métodos de aprendizaje de modelos gráficos probabilísticos, su relación con los EDA, así como las principales propuestas secuenciales y paralelas encontradas en la revisión bibliográfica. Caracterizar los algoritmos de aprendizaje estructural de redes Bayesianas que basan su funcionamiento en la detección de independencias en el entorno de los EDA.
2. Precisar las potencialidades de los algoritmos que aprenden estructuras simples y multi-conectadas a partir de pruebas de independencia + optimización de métrica.
3. Crear y evaluar esquemas EDA basados en pruebas de independencia para dominio discreto y continuo.
4. Crear y evaluar algoritmos paralelos para el aprendizaje de distribuciones basadas en pruebas de independencia y extender su utilización a EDA.
5. Caracterizar y evaluar los modelos descentralizados, el de islas y el celular, como herramientas algorítmicas para aumentar la eficiencia evaluativa y en tiempo de los EDA.

La tesis representa un aporte al desarrollo de los Algoritmo de Estimación de Distribuciones con la propuesta de nuevos algoritmos evolutivos que permitan resolver problemas prácticos eficientemente. Los resultados científicos de la investigación son los siguientes:

1. Creación de la clase de algoritmos EDA Bayesianos basados en restricciones (detección de independencias), CBEDA (capítulo 2).
   a) Se crean y estudian experimentalmente dos nuevos EDA basados en esquemas de aprendizaje híbrido ($\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{TPDA}}$), *detección de independencias + optimización de métrica* para reducir el número de evaluaciones de los algoritmos que utilizan sólo optimización de métricas en los problemas estudiados.
   b) Se caracteriza el comportamiento numérico de los CBEDA creados, utilizando el método de las B-funciones.
2. Se crean y estudian experimentalmente tres nuevos EDA paralelos ($\text{pCBEDA}_{\text{LPA}}$, $\text{pCBEDA}_{\text{MMHC}}$ y $\text{pCBEDA}_{\text{TPDA}}$) basados en esquemas de aprendizaje híbrido y aumentando la eficiencia en tiempo (capítulo 3).
3. Se crean dos esquemas descentralizados (*distribuido* y *celular*) en EDA para la optimización de funciones discretas y continuas (capítulo 4).
   a) Los esquemas creados son más eficientes que la versión centralizada con respecto al número de evaluaciones.
   b) El algoritmo de islas para el UMDA es eficiente en tiempo con ganancias de velocidad superlineales.

c) Se muestra una explicación teórica sobre la eficiencia evaluativa del modelo de islas, basado en UMDA y con selección por truncamiento.

La novedad e importancia de la tesis queda reflejada en los resultados antes citados. Por primera vez se proponen EDA basados en los algoritmos de aprendizaje TPDA [26] y MMHC [163], además de estar paralelizados, lo cual no es solo importante en EDA sino también para el aprendizaje automatizado. Otro resultado novedoso es la propuesta de dos versiones paralelas de PADA. Todos los algoritmos propuestos se encuentran en el estado del arte de EDA y de aquí su importancia para abordar problemas de manera más eficiente que otros algoritmos.

La tesis consta de introducción, cuatro capítulos, conclusiones, recomendaciones, bibliografía y anexos.

En el capítulo 1 de la tesis se presenta una revisión del estado del arte de los algoritmos EDA paralelos y secuenciales. Además, se enuncian los conceptos y definiciones utilizadas y se ofrece una valoración sobre el estado de las investigaciones y aplicaciones, tanto en EDA secuenciales como paralelos.

En el capítulo 2 se presenta un estudio sobre los algoritmos de aprendizaje de redes Bayesianas basados en pruebas de independencia y su utilización en EDA. Se propone el $\text{CBEDA}_{\text{MMHC}}$, un EDA sustentado en el algoritmo de aprendizaje MMHC. Se introducen modificaciones en el proceso de aprendizaje del algoritmo TPDA y se propone el $\text{CBEDA}_{\text{TPDA}}$. Por último se muestra un estudio empírico para evaluar las características y potencialidades de las nuevas propuestas.

En el capítulo 3 se proponen las versiones paralelas de los algoritmos $\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{TPDA}}$ estudiados en el capítulo 2. Se introducen dos implementaciones paralelas del algoritmo $\text{CBEDA}_{\text{LPA}}$, basadas en el aprendizaje de poliárboles. El capítulo 4 propone dos modelos descentralizados, el de islas y el celular, y su aplicación al contexto de los EDA.

Finalmente se presentan las conclusiones y recomendaciones de la investigación. En el anexo A se detallan las funciones utilizadas en los experimentos. El anexo B enumera las publicaciones del autor relacionadas con la investigación.

CAPÍTULO 1

CONTEXTO: ALGORITMOS DE ESTIMACIÓN DE DISTRIBUCIONES

1.1. Introducción

Los EDA son optimizadores que engloban a un conjunto de técnicas que pertenecen a la clase de los EA. La principal característica de estos algoritmos es la utilización de modelos probabilísticos para detectar las relaciones de independencia entre las variables del problema a resolver. El algoritmo EDA es una propuesta teórica, cuya variante práctica la constituyen los Algoritmos con Distribución Factorizada (FDA). Existe una estrecha relación entre la teoría de los modelos gráficos probabilísticos y los FDA, especialmente con las redes Bayesianas, redes Gaussianas y las redes de Markov.

No obstante a la posibilidad que brinda la teoría de la factorización y su aplicación a los EDA, en la práctica, la estimación del modelo probabilístico es una tarea costosa desde el punto de vista computacional. Las técnicas paralelas brindan la posibilidad de abordar problemas complejos en tiempos relativamente cortos y así se convierten en candidatas para reducir el costo en tiempo de los EDA.

En este capítulo se introducen las notaciones y conceptos relacionados con el algoritmo EDA, entre ellos el término de Algoritmo Evolutivo de Bajo Costo. Se presentan los distintos tipos de aprendizaje y simulación, existente en las redes Bayesianas. Por último se presenta una revisión detallada sobre los principales algoritmos (secuenciales y paralelos) del estado del arte así como sus características y se presentan algunas aplicaciones de los EDA.

1.2. Algoritmos de estimación de distribuciones

Los EDA se introducen por primera vez en el campo de la computación evolutiva en 1996 [104]. Se crearon para sustituir los operadores de cruzamiento y mutación por operadores de estimación de la distribución de probabilidad y muestreo. Recientes trabajos en esta área demostraron la utilidad del operador de mutación en este tipo de algoritmo como un operador de *variación probabilística* [150, 118]. De forma general los EDA utilizan una población seleccionada para estimar la distribución de probabilidad y a partir de esta se generan los puntos que formarán la nueva población. Así, las relaciones de independencia entre las variables quedan reflejadas, de forma explícita, mediante el modelo probabilístico del conjunto seleccionado.

A continuación se detallan un conjunto de definiciones y notaciones para un mejor entendimiento de los EDA.

Notación y definiciones

Las variables se denotan con una letra mayúscula (A, V_i) y el estado o valor, por la misma letra en minúscula (a, v_i). Los conjuntos de variables son representados por letras mayúsculas en negrita ($\mathbf{Z}$, $\mathbf{Pa}_i$) y la correspondiente letra minúscula en negrita para especificar los posibles valores que toma el conjunto.

Esta tesis hace énfasis en la solución de problemas de optimización definidos de la siguiente manera:

$$x_{opt} = arg\,max_{x \in \mathscr{D}^n} f(x)$$

Donde $\mathbf{x} = (x_1, x_2, \cdots x_n)$ denota un vector discreto o continuo de variables aleatorias. En el caso discreto, cada $x_i \in \{0, 1, \cdots, r_i\}$, en otras palabras, la variable x_i puede tomar $r_i + 1$ valores. En el caso continuo cada $x_i \in [a_i, b_i]$, la variable x_i toma, de forma continua, todos los posibles valores comprendidos en el intervalo $[a_i, b_i]$. Este problema consiste en encontrar el punto máximo de la función $f(\mathbf{x}) \Rightarrow \mathscr{R}$. En ambos dominios $i \in \{1, 2, \cdots, n\}$.

Los algoritmos de optimización estocásticos se expresan por un modelo, donde a cada configuración $x = [x_1, x_2, \cdots, x_n]$ le corresponde una distribución $p(x) = p(x_1, x_2, \cdots, x_n) = P(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n)$, según la distribución $P(X)$.

Conceptualmente se define el esquema de EDA según se muestra en el algoritmo 1.1. Primero, se inicializa el contador de generaciones a uno y se crea una *población inicial* con N puntos, generados aleatoriamente. A continuación, y mientras no se cumpla el *criterio de parada* (encontrar el óptimo, cantidad de generaciones, etc.), se evalúa la población y se seleccionan M puntos de acuerdo a un método de selección (generalmente los mejores, *selección por truncamiento*) y con ellos se crea el *conjunto seleccionado CS*. Posteriormente, se estima la distribución de probabilidad del conjunto seleccionado (CS) y se genera la nueva población. Por último, se incrementa el contador de generaciones y el algoritmo vuelve a iterar.

Algoritmo 1.1 EDA Simple

Poner $t \leftarrow 1$
Generar $N \gg 0$ puntos de forma aleatoria
mientras no se cumpla el criterio de parada **hacer**
    Evaluar la población en la función $f(x)$
    De acuerdo a un método de selección, construir un conjunto CS de M puntos
    Estimar la distribución del conjunto seleccionado $p^{CS} = p(x,t)$ a partir de CS
    Generar N nuevos puntos a partir de $p(x, t+1) \approx p^{CS}(x,t)$
    Poner $t \leftarrow t + 1$
fin mientras

El algoritmo EDA es teórico, producto a que el cálculo de todos los parámetros necesarios para especificar la distribución de probabilidad es intratable. Por ejemplo, para almacenar la distribución de probabilidad de n variables aleatorias binarias se necesitan $2^n - 1$ parámetros (probabilidades).

Producto a esta dificultad es que surge el algoritmo con distribución factorizada con el objetivo de hacer de los EDA algoritmos tratables. Hay que destacar que en la literatura varios autores asumen a los EDA como algoritmos prácticos [132, 128, 72, 150] y se fundamenta en la existencia de un algoritmo que utiliza factorizaciones fijas y que lleva el nombre de FDA [103]. De la misma forma, se habla de EDA como la clase de los algoritmos evolutivos basados en estimación y simulación de distribuciones.

Un término que se relaciona con EDA es el de *Algoritmo Evolutivo de Bajo Costo* (LCEA) [113, 116] y tiene en cuenta un conjunto de rasgos que deben presentar los algoritmos evolutivos:

- aprendizaje y utilización de la estructura probabilística del problema,
- aprender ”apropiadas" funciones de evaluación,
- realizar evaluación parcial de los individuos,
- utilización de técnicas paralelas y distribuidas.

Estas directivas están encaminadas a crear algoritmos evolutivos eficientes. De forma similar, Goldberg redefinió este término como Algoritmos Genéticos Competentes (*Competent Genetic Algorithms*) [51]. Esta investigación se dirige al primero y al último punto de la lista de estrategias de los LCEA, al aprendizaje de la estructura probabilística del problema y a la utilización del paralelismo.

La primera línea de investigación refiere que la detección y utilización de las interacciones más importantes entre las variables del problema forman la clave para lograr un muestreo eficiente del espacio de soluciones. La meta del último punto está relacionada a la utilización de las arquitecturas paralelas y distribuidas para: (a) reducir el costo computacional de cada paso del EDA (evaluación, aprendizaje, simulación, etc.) y (b) reducir el número de evaluaciones de la función objetivo.

Existe una estrecha relación entre la clase de los algoritmos EDA y los algoritmos LCEA. Los primeros constituyen la base del LCEA, donde se incluyen un conjunto de estrategias para disponer de un algoritmo eficiente y escalable.

1.3. Modelos gráficos probabilísticos

Los Modelos Gráficos (GM) constituyen herramientas que permiten representar distribuciones de probabilidad conjunta. Los Modelos Gráficos Probabilísticos (PGM) son grafos en los cuales los nodos representan variables aleatorias y los arcos representan relaciones de dependencia condicional. Los mismos proveen una forma compacta de representar la distribución de probabilidad. El modelo gráfico tiene cuatro componentes fundamentales [17]: la semántica, la estructura, la implementación y los parámetros.

Para los GM existen una variedad de semánticas posibles e incluyen los modelos dirigidos (redes Bayesianas y redes Gaussianas), los modelos no dirigidos (redes de Markov), entre otros. La estructura está enfocada a las relaciones de independencia presentes en el grafo, con la existencia o no, de arcos entre los nodos y la dirección que tienen los mismos. Una vez fijada la estructura, la dependencia entre las variables se puede implementar utilizando árboles de decisión, tablas de probabilidades

condicionales (CPT), etc. Los parámetros del modelo quedan almacenados en forma de probabilidades condicionales.

En esta tesis se utilizan dos semánticas, las redes Bayesianas y las redes Gaussianas. Para representar las probabilidades se utilizan las tablas de probabilidades condicionales y la matriz de covarianza en el caso continuo. Las variables aleatorias toman valores discretos o continuos según el problema de optimización.

1.3.1. Redes Bayesianas

Una red Bayesiana es un tipo de GM que utiliza Grafos Acíclicos Dirigidos (DAG), por lo que toma en consideración la dirección de los arcos. Una red Bayesiana se define mediante el par $\langle \mathscr{G}, P \rangle$, donde $\mathscr{G}$ es un grafo que representa las relaciones de dependencia entre las variables y P es la factorización de la distribución de probabilidad representada por $\mathscr{G}$.

Formalmente se define una red Bayesiana sobre un conjunto, $V = \{V_1, \ldots, V_n\}$, de variables aleatorias. La factorización de la probabilidad conjunta puede expresarse como:

$$P(\mathscr{V}) = P(V_1, V_2, \cdots, V_n) = \prod_{i=1}^{n} P(V_i \mid Pa_i) \tag{1.1}$$

La expresión 1.1 permite definir una red Bayesiana con la condición de Markov, cada variable (V_i) es independiente de cualquier subconjunto de las variables no descendiente de ella, condicionado en su conjunto de padres (Pa_i).

La semántica de la red Bayesiana demanda una clara correspondencia entre la topología del grafo y las relaciones de dependencias representadas por el DAG. Esta correspondencia se sustenta en el criterio de d-separación [123, 124] y considera la dirección de los arcos en el grafo.

Una definición importante es la de red Bayesiana fiel:

Definición 1. *Una red Bayesiana $\langle \mathscr{G}, P \rangle$ satisface la* condición de fidelidad *si P contiene solo las independencias que pueden ser representadas en el* DAG *$\mathscr{G}$ [155]. Se le denomina a esta red Bayesiana* red fiel.

Esta definición es muy importante para los algoritmos de aprendizaje utilizados en el capítulo 2, pues relacionan la fidelidad de la red a una distribución de probabilidad. Cabe destacar que existen distribuciones P para las cuales no existe una red Bayesiana fiel a la misma, sin embargo, estas distribuciones clasifican como ”raras” [94]. De la misma forma, puede existir más de un grafo para representar la misma distribución de probabilidad P.

Ahora se puede definir el problema del aprendizaje de la estructura de la red Bayesiana.

Definición 2. *Sea P una distribución fiel y $\mathscr{D}$ una muestra estadística generada a partir de P. El problema del aprendizaje de la estructura de la red Bayesiana dado $\mathscr{D}$ consiste en inducir el grafo $\mathscr{G}$ tal que $\langle \mathscr{G}, P \rangle$ es una red Bayesiana fiel.*

Teorema 3. *En una red Bayesiana fiel BN $\langle \mathscr{G}, P \rangle$ sobre el conjunto de variables $\mathscr{V}$ existirá un arco entre cualquier par de nodos $X, Y \in \mathscr{V}$, si y sólo si se tiene $Dep_P(X, Y \mid Z)$, para todo $Z \subseteq \mathscr{V}$ [155].*

Varios de los algoritmos basados en restricciones para el aprendizaje de redes Bayesianas utilizan el teorema anterior. Estos estiman, a partir de los datos, si se tienen ciertas relaciones de independencia entre las variables del problema. Para esto se utilizan pruebas estadísticas o pruebas basadas en la teoría de la información [49, 26]. El subíndice P será eliminado de las expresiones.

A continuación se define el concepto de independencia condicional.

Definición 4. *Dos variables X e Y son* condicionalmente independientes *dado Z con respecto a una distribución de probabilidad P, y se denota como $Ind_P(X, Y \mid Z)$, si $\forall x, y, z$ donde $P(Z = z) > 0$,*

$$P(X = x, Y = y \mid Z = z) = P(X = x \mid Z = z) P(Y = y \mid Z = z)$$

ó

$$P(X, Y \mid Z) = P(X \mid Z) P(Y \mid Z)$$

Para la implementación de las pruebas de independencia condicional se calcula el estadístico G^2 [156], bajo la hipótesis nula de independencia condicional. Sea N_{ijk} el número de veces que en los datos $X_i = x_i, X_j = x_j, X_k = x_k$. De la misma forma se define, N_{ik}, N_{jk}, y N_k. Entonces, el estadístico G^2 se define como [156]:

$$G^2 = 2 \sum_{ijk} N_{ijk} \ln \frac{N_{ijk} N_k}{N_{ik} N_{jk}} \tag{1.2}$$

El estadístico G^2 se distribuye asintóticamente como χ^2 con apropiados grados de libertad. Asumiendo que no existen ceros estructurales, el número de grados de libertad es:

$$df = (\mid C(X_i) \mid -1)(\mid C(X_j) \mid -1) \prod_{X_l \in X_k} (\mid C(X_l) \mid) \tag{1.3}$$

donde $C(X)$ es la cardinalidad de la variable X.

La prueba χ^2 retorna un *p-value* que corresponde a la probabilidad de rechazar falsamente la hipótesis nula dado que esta es verdadera. Si el *p-value* es menor que un nivel de significación ε (en esta investigación $\varepsilon = 0{,}05$) la hipótesis nula es rechazada. Si la hipótesis de independencia no puede ser rechazada, entonces se acepta.

1.3.2. Aprendizaje de redes Bayesianas

Una característica de las redes Bayesianas es que pueden ser aprendidas a partir de un conjunto de datos (población en el entorno de los EDA). Existen dos técnicas fundamentales para el aprendizaje de una red Bayesiana: aprendizaje basado en restricciones (constraint based learning), también conocidos

como algoritmos que detectan independencias y el aprendizaje basado en optimización de métricas (search-and-score based learning), conocidos como métodos de puntuación.

Existen múltiples propuestas para el aprendizaje automático de redes Bayesianas [59, 66, 26, 163]. Los algoritmos que aprenden redes Bayesianas tienen que llevar a cabo dos tareas fundamentales: (1) realizar un aprendizaje estructural de la red y a partir de este, (2) estimar los parámetros (aprendizaje paramétrico) representados mediante probabilidades condicionales.

El aprendizaje de una red Bayesiana a partir de los datos es un problema NP-Completo [27, 28]. A continuación se analizan, de forma breve, las alternativas existentes para el aprendizaje de redes Bayesianas. Se refiere al lector al capítulo 2 para más detalles sobre el aprendizaje basado en restricciones y su utilización en EDA.

Aprendizaje paramétrico

Independientemente del método de aprendizaje estructural que se utilice, existe una forma fácil y común de calcular las probabilidades condicionales a partir de la estructura de la red Bayesiana. El método se conoce como cálculo de frecuencias relativas

$$p(x_i \mid Pa_i) = \frac{n(x_i, Pa_i)}{n(Pa_i)}$$

donde $n(x_i, Pa_i)$ representa el número de casos en los datos para los cuales la variable X_i y sus padres toman simultáneamente los valores x_i y Pa_i, y $n(Pa_i)$ representa el número de casos donde los padres de X_i toman los valores Pa_i. Este método se corresponde con la utilización de un estimador por máxima verosimilitud [22].

Otra forma utilizada para estimar las probabilidades condicionales de la red es la estimación Bayesiana [60]. El objetivo es eliminar el problema del sobreajuste de los datos, que no es más que la asignación, erróneamente, de un valor igual a cero a determinadas probabilidades. En el entorno de los EDA este estimador se ha interpretado como un operador de mutación [91].

Aprendizaje basado en optimización de métricas

Este método intenta identificar una red Bayesiana que maximiza una función de costo e identifica la estructura que mejor se ajusta a los datos. Generalmente emplean técnicas de búsqueda local como: algoritmos ávidos, escaladores de colinas, los EDA [135], entre otros. Ejemplos de métricas utilizadas en este tipo de algoritmo son: el Criterio de Información Bayesiana (BIC) [147], el Criterio de Información de Aikaike (AIC) [3] y la métrica Bayesiana-Dirichlet (BDe) [61].

Estos métodos son útiles cuando se intenta aprender el modelo a partir de conjuntos de datos con pocas instancias, cuando no están especificados todos los valores de los rasgos que componen los patrones y para inferir modelos que no se obtienen mediante los métodos basados en restricciones [108]. La principal desventaja es que no aprenden modelos donde existen variables ocultas (latentes).

En esta tesis se utiliza la métrica BIC en el proceso de orientación de la red Bayesiana. A partir de la estructura S y una base de datos D (conjunto seleccionado), la BIC se expresa como:

$$BIC(S,D) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk} \cdot log\frac{N_{ijk}}{N_{ij}} - \frac{1}{2} \cdot log(N)\sum_{i=1}^{n} q_i(r_i - 1) \tag{1.4}$$

donde n es el número de variables de la red Bayesiana, r_i es la cardinalidad de cada variable x_i, q_i es la combinación de valores que toman los padres de cada variable Pa_i, N_{ij} es el número de casos en D para los cuales Pa_i toma el valor j, y N_{ijk} es el número de casos en D para los cuales la variable X_i toma su valor $k-ésimo$ y sus padres Pa_i toman su valor j-ésimo.

Una propiedad importante de esta métrica es que se descompone, es decir, se calcula como una suma de las BIC locales de cada variable ($BIC(i,S,D)$).

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

donde

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

Aprendizaje basado en restricciones

Los algoritmos que utilizan este método estiman a partir de los datos si existen ciertas relaciones de dependencias entre las variables. Típicamente, estas estimaciones se realizan empleando medidas estadísticas o mediante la teoría de la información. Dos de los métodos propuestos para el aprendizaje de la estructura de la red Bayesiana y se fundamentan en la detección de independencias condicionales, son el algoritmo PC [155, 157, 156] para el aprendizaje de redes generales y los algoritmos para el aprendizaje de poliárboles presentados por Acid [1].

El esquema de aprendizaje que se comienza a imponer es la utilización de técnicas híbridas. Primero se construye un esqueleto utilizando un procedimiento basado en restricciones y posteriormente se aplica un método de puntuación para orientar las aristas, aplicado sobre un espacio reducido las posibles redes Bayesianas [108, 151, 20, 21, 162, 163].

1.3.3. Simulación de redes Bayesianas

La simulación de una red Bayesiana consiste en generar un vector de las variables del problema, utilizando el modelo probabilístico que representa la estructura de la red.

Dentro de los algoritmos más empleados se encuentran los simuladores de Montecarlo, específicamente, el Muestreo Lógico Probabilístico (PLS) [63]. El esquema de funcionamiento del PLS es como sigue: se ordenan las variables siguiendo un orden ancestral, los padres Pa_i de cualquier variable X_i se encuentran por delante en el orden generado. Posteriormente, se calcula la probabilidad condicional de X_i dado el conjunto de padres, $p(x_i \mid Pa_i)$. Con la probabilidad estimada se genera el valor de la variable X_i. Este procedimiento se repite para todos los individuos a generar.

1.3.4. De los modelos gráficos probabilísticos a los EDA

La relación que existe entre los modelos gráficos y los EDA es bastante estrecha, los primeros se utilizan para representar el modelo probabilístico del conjunto seleccionado del algoritmo EDA. Según la información disponible sobre el problema, que casi nunca se tiene, puede prefijarse el tipo de modelo probabilístico que incluye modelos con total independencia, bivariados, trivariados o modelos generales. De igual forma, se tienen EDA que utilizan un modelo probabilístico a priori para realizar aprendizaje paramétrico a partir de los datos [12, 99, 103].

Cuando los modelos gráficos se utilizan como generadores se muestrean las nuevas soluciones que se incorporan a la población global del EDA. El método más extendido es el PLS y en el caso de los EDA Markovianos se ha propuesto el muestreo de Gibbs [138, 139]. Otro de los algoritmos que se emplean para generar nuevas soluciones es el de las k configuraciones más probables [109, 166].

1.4. EDA secuenciales

En esta sección se presentan las principales contribuciones en EDA secuenciales que sirven de motivación para esta investigación. Los algoritmos son clasificados según el tipo de aprendizaje que realizan. También se efectúa una breve clasificación en cuanto al orden de las dependencias consideradas.

1.4.1. Clasificación de los EDA

Los algoritmos EDA se clasifican teniendo en cuenta el orden de las dependencias en las variables del problema [73, 129] y se dividen en tres grandes grupos: los que asumen total independencia (UMDA [99], PBIL [12] y el cGA [58]), los que utilizan modelos bivariados (MIMIC [34], MIMIC_{C} [73, 75], BMDA [131], PADA_{t2} [150]), y los que no ponen restricciones sobre el modelo (EBNA_{BIC} [39, 73], BOA [128], LFDA , ECGA [146], EGNA_{BGe} [73, 75] y MMHCEDA [89]).

Otra forma de clasificación de los EDA consiste en analizar el tipo de aprendizaje que realizan [138].

- **Aprendizaje paramétrico.** Los algoritmos de esta categoría son el UMDA [99], el PBIL [12], el cGA [58] y el FDA [100].
- **Aprendizaje estructural y paramétrico.** Pertenecen la mayoría de los EDA que utilizan algoritmos de aprendizaje automático para estimar el modelo.
    - **Optimización de métrica.** Algoritmos como el BOA [128], LFDA [102], ECGA [146], EBNA_{BIC} [73], SPADA [150], entre otros.
    - **Detección de independencias.** Sus principales exponentes son PADA_{t2} [150], EBNA_{PC} [73], entre otros.
    - **Detección de independencias + optimización de métrica.** De los EDA más extendidos se tienen al PADA_{p3} [151, 152, 150], BMDA [131], MIMIC [34], y el único que aprende redes generales, el MMHCEDA [89].

1.4.2. Estado del arte en EDA secuenciales

Esta sección muestra los principales algoritmos del estado del arte que son objeto de estudio y motivación de los resultados de esta tesis. Los algoritmos se presentan teniendo en cuenta el dominio de definición de las variables del problema (discreto o continuo).

Dominio discreto

El Algoritmo con Distribución Marginal Univariado (UMDA) fue propuesto por [99], extendiendo el trabajo previo desarrollado en [107]. Este es el algoritmo más simple de todos los que forman la familia de los EDA. Cada distribución marginal univarada se estima a partir de las frecuencias marginales para cada variable. Es un algoritmo eficiente y se comporta de forma excelente en problemas lineales donde no existe interdependencia fuerte entre las variables, aunque es capaz de optimizar funciones con interacciones entre las variables [119]. Dentro del grupo de algoritmos que asumen independencia entre las variables y que tienen un comportamiento similar al UMDA, se encuentran el PBIL [12] y el algoritmo Genético Compacto (cGA) [58]).

El Algoritmo de Maximización de la Información Mutua para la Clasificación de Entradas (MIMIC) fue propuesto por De Bonet y col. [34]. En cada generación, MIMIC busca la mejor permutación π entre las variables para estimar la distribución de probabilidad conjunta que tiene una estructura de cadena. Para determinar la mejor permutación se utiliza la distancia de Kullback-Leiber y se detecta la cadena que está más próxima a la distribución del conjunto seleccionado.

El Algoritmo con Distribución Factorizada Basada en Poliárboles (PADA) [150] utiliza el poliárbol como modelo gráfico. La construcción de la factorización se realiza en dos pasos: primero se aprende un esqueleto del poliárbol utilizando pruebas de independencia. Después se utiliza la información sobre las dependencias marginales y condicionales para darle dirección a algunas aristas y transformarlas en arcos. Al resto de las aristas que quedan sin orientar se les aplica un proceso de optimización de métrica para completar el direccionamiento.

El Algoritmo de Optimización Bayesiana (BOA) [128, 126, 127] asume interacciones generales entre las variables y representa el modelo de probabilidad conjunta mediante una red Bayesiana. BOA utiliza la métrica BDe para determinar cuál es la red que mejor se ajusta a los datos. Se utiliza un parámetro k (número máximo de padres para cualquier nodo) para controlar y limitar la búsqueda sobre el espacio de todas las posibles estructuras. Otra propuesta de algoritmo de optimización Bayesiana es el Algoritmo Evolutivo con Estimación de Redes Bayesianas (EBNA) [39, 73]. El aprendizaje en EBNA se realiza por diferentes métodos como el aprendizaje por optimización de métricas (EBNA_{BIC}, $\text{EBNA}_{\text{k2+pen}}$) y la detección de independencias (EBNA_{PC}). Similar al EBNA y al BOA es el Algoritmo Evolutivo con Aprendizaje de Distribuciones Factorizadas (LFDA) [102].

Santana y col. [144, 145] se emplean las mezclas de árboles para el aprendizaje de la distribución de búsqueda del EDA, específicamente, el Algoritmo Evolutivo con Factorización de Mezclas de Árboles (MT-FDA). En [138] se extienden estos resultados proponiendo un EDA basado en mezclas de distribuciones con aproximaciones de Kikuchi [167, 168].

En [121] se propone el Algoritmo Evolutivo con Distribución Factorizada y Aprendizaje (FDA-learning). El mismo representa factorizaciones válidas[1] con cliques que se solapan. Este algoritmo construye una red Markoviana utilizando como base un algoritmo para el aprendizaje de grafos cordales[2] (grafos triangulares) propuesto en [36]. En [138, 137] se recomienda la utilización de las aproximaciones de Kikuchi [167, 168] para la construcción de factorizaciones inválidas. El algoritmo resultante se nombra Algoritmo Evolutivo con Estimación de Distribuciones con Redes de Markov (MN-EDA). Este algoritmo es una extensión del Algoritmo Evolutivo con Distribución Factorizada basado en Redes de Markov (MN-FDA) [136, 138] que utiliza factorizaciones válidas al igual que el FDA-learning.

Gámez y col. [46, 47] proponen un nuevo EDA basado en la estimación de redes de dependencias (EDNA). El algoritmo considera solo dependencias bivariadas para estimar la distribución de búsqueda del conjunto seleccionado. El EDNA se comparan con EDA Bayesianos, EBNA y hBOA, concluyendo que los resultados son comparables.

Dominio continuo

El Algoritmo con Distribución Marginal Univariado para dominio continuo (UMDA_C) fue propuesto en [73, 75]. En cada generación el algoritmo asume que las variables son independientes y que siguen una distribución normal (otras distribuciones se pueden tener en cuenta). Los dos parámetros fundamentales que estima el algoritmo en cada generación t y para cada variable son: la media, $\mu_i(t)$, y la desviación estándar, $\sigma_i(t)$. Siguiendo un esquema similar al UMDA se encuentra el algoritmo con Aprendizaje Incremental Basado en Poblaciones para dominio continuo (PBIL_C) [148].

El Algoritmo de Maximización de la Información Mutua para la Clasificación de Entradas para dominio continuo (MIMIC_C) [73, 75] no es más que una extensión del algoritmo MIMIC [34] con la diferencia que las variables toman valores reales. La idea es similar a la que sigue el algoritmo discreto, fijando el modelo a los datos empíricos y analizando solo las relaciones que existen entre pares de variables, de la misma forma que UMDA_C.

El Algoritmo de Estimación de Distribución Normal Multivariado (EMNA) [71] utiliza una función de densidad normal multivariada para aprender la factorización de los individuos seleccionados. La estimación del vector de medias y la matriz de covarianza se realiza a través de un estimador de máxima verosimilitud. Alternativas a este algoritmo (se conoce como $\text{EMNA}_\text{global}$) son el EMNA_a y el EMNA_i, ambos generan un solo individuo. El primero es adaptativo, incorpora el individuo si es mejor que el peor de la población. El segundo es incremental, adiciona el individuo a la población.

Otra propuesta de EDA para dominio continuo lo constituye el Algoritmo con Estimación de Redes Gaussianas (EGNA). El mismo utiliza aprendizaje y simulación de redes Gaussianas. Una de las variantes propuestas es el EGNA_BGe, donde la inducción del modelo se realiza mediante un método de puntuación. Un esquema similar sigue el EGNA_BIC con la diferencia que emplea la métrica BIC para

[1] Se puede construir un árbol de cliques

[2] Cualquier ciclo de longitud mayor o igual que cuatro tiene una cuerda

dominio continuo. Por último se propone el $EGNA_{EE}$ (Algoritmo con Estimación de Redes Gaussianas con Exclusión de Arcos) que utiliza detección de independencias para construir la red Gaussiana.

Otra propuesta es el algoritmo PolyEDA [53] que no es más que la combinación de algoritmos de estimación de distribuciones y restricciones con desigualdades lineales. Este algoritmo fue propuesto para la solución de problemas con restricciones.

1.5. EDA paralelos

La investigación en las tecnologías de las comunicaciones y la computación, en las últimas décadas, aportó nuevas herramientas de software y hardware a la comunidad científica en general. Específicamente, los avances en las arquitecturas de los microprocesadores, las redes de computadoras y la creación de nuevos software ha permitido a los investigadores proponer nuevos algoritmos capaces de abordar problemas de mayor complejidad y demanda de recursos.

La disponibilidad de potentes recursos computacionales como los conglomerados de computadoras, las arquitecturas de varios núcleos, las máquinas paralelas, entre otros, propicia el desarrollo de aplicaciones paralelas y distribuidas que hacen uso de estos recursos. Estas aplicaciones reducen el tiempo de cómputo, el espacio de memoria utilizada, mejoran la calidad de las soluciones al problema que se resuelve y permiten manejar problemas de mayores dimensiones comparados con los que se abordan en las propuestas secuenciales.

Los EA y específicamente los EDA tienen un funcionamiento intrínsecamente paralelo por lo que se convierten en excelentes candidatos para ser paralelizados. En la literatura se encuentran diferentes propuestas paralelas para esta familia de algoritmos que mejoran los resultados (tanto evaluativo como en tiempo) de la propuesta secuencial. Los trabajos en esta área utilizan dos técnicas fundamentales, (1) la distribución espacial de la población de individuos con intercambio de información entre los nodos computacionales y (2) los que siguen un esquema de funcionamiento similar a la propuesta secuencial pero que se centran en la reducción del tiempo de cómputo para abordar problemas de mayores dimensiones.

1.5.1. Niveles de paralelismo en EDA

Según el esquema general de funcionamiento de un algoritmo EDA (pseudo-código 1.1), el mismo puede ser paralelizado a diferentes niveles:

- (L) - Nivel de aprendizaje (o estimación)
- (S) - Nivel de muestreo (o simulación)
- (P) - Nivel de población
- (F) - Nivel de evaluación de la función objetivo

En general, el aprendizaje de una red Bayesiana a partir de datos es un problema NP-Completo [27, 28], debido a que se requiere de un esfuerzo computacional exponencial. Muchos de los algoritmos de aprendizaje son exponenciales en el número máximo de padres de la red. Esto implica

que el aprendizaje es el paso de los EDA que mayor tiempo consume y uno de los más promisorios para la aplicación de técnicas paralelas [95].

El muestreo es otro de los pasos que se paraleliza; la generación de nuevos individuos se realiza de una manera naturalmente paralela. El método de muestreo más popular en las implementaciones de EDA discretos es el algoritmo PLS [63]. Para problemas con grandes poblaciones y gran número de variables la simulación es costosa.

A nivel de población se encuentra la técnica que hace énfasis en la estructura espacial de la misma. Una población global está definida *virtualmente* por un conjunto de poblaciones locales que interactúan entre sí. La fuerza y la frecuencia de las interacciones, y los tamaños de las sub-poblaciones definen el esquema de paralelización. Uno bien estudiado es el modelo de islas, que son grupos de individuos semi-autónomos, o sub-poblaciones con asociaciones débiles con las islas vecinas. Esta asociación está centrada en la migración de individuos de una isla a otra. Esta técnica admite una fácil paralelización que ha sido extensamente investigada en el espacio de los EA [7, 25]. Los resultados existentes son fácilmente ampliados al dominio de los EDA [2, 87, 69]. Recientemente se introdujo una nueva clase de algoritmos EDA que se caracteriza por tener una asociación fuerte entre las sub-poblaciones vecinas [86, 5]. Este modelo se conoce como cEDA y fue investigado en el capítulo 4.

La evaluación de la función de aptitud es otro de los componentes importante del costo de un EDA, particularmente en aplicaciones prácticas. Sin embargo, no se tienen nuevas ideas en EDA que no se hayan encontrado en la paralelización de otros algoritmos evolutivos.

La lista de niveles de paralelismo no es exhaustiva. Existen otros operadores que son paralelizados en EDA; por ejemplo la selección. Además, la investigación en EDA está aún en sus inicios y se espera que se desarrollen nuevos métodos que necesiten la utilización de técnicas paralelas. Por ejemplo, un algoritmo espacialmente estructurado puede combinarse con un modelo maestro/trabajador para la evaluación de la función objetivo y/o la computación del modelo probabilístico. Para más detalles, en la aplicación de los niveles de paralelismo y una revisión específica sobre EDA paralelos, referimos al lector a los trabajos de Madera y Mendiburu[86, 95].

1.5.2. Estado del arte en EDA paralelos

En el campo de los EDA paralelos se desarrollaron diferentes algoritmos interesantes. Lozano y col. [81] proponen dos versiones paralelas para un algoritmo que emplea los modelos gráficos probabilísticos en el campo combinatorio (EBNA_{BIC}). Este enfoque requiere que la comunicación se establezca mediante estructuras de datos compartidas que son accedidas en un ambiente multi-hilo, por lo que no es directamente aplicable en el contexto de las computadoras con memoria distribuida (incluyendo los clústeres de computadoras).

Relevantes son los trabajos desarrollados por Ocenasek [111, 112] donde se proponen dos métodos de paralelización para otro de los algoritmos pertenecientes a la familia de los EBNA, el algoritmo BOA. La idea central del pBOA [111] es la construcción de la red Bayesiana de tal forma que cada procesador inserta arcos independientemente de los demás, y así en cada generación se realiza

una permutación aleatoria de las variables del problema con el objetivo de que el grafo resultante se mantenga acíclico. En el caso del algoritmo dBOA [112] el aprendizaje de la red Bayesiana se desarrolla en un ambiente distribuido donde el procesador mantiene una copia de la población seleccionada y realiza la estimación del modelo probabilístico. Para mantener la red acíclica se utiliza una idea similar al pBOA. La generación de nuevos individuos se desarrolla de forma distribuida, lo que permite que cada proceso genere una porción de la población global y se intercambie entre todos los procesos.

Mendiburu y col. [96] proponen varias implementaciones paralelas de algoritmos EDA para dominio discreto y continuo. Todas utilizan dos interfaces de programación paralela distintas (API): Pase de Mensajes (MPI) [42] y POSIX threads [23]. Esto permite emplear un ambiente combinado de programación multi-hilos con pase de mensajes en un entorno distribuido. El primer algoritmo que se implementa es el $\text{pEBNA}_{\text{BIC}}$ donde la funcionalidad secuencial es preservada y se utiliza el procesamiento paralelo para acelerar algunas porciones del algoritmo. Este se centra en el algoritmo EBNA_{BIC}, se extienden los trabajos de [81, 111, 112] y permite el aprendizaje de la red Bayesiana sin tener en cuenta restricciones en el orden de las variables del problema. El segundo algoritmo propuesto es el pEBNA_{PC} cuya funcionalidad es similar al anterior pero utiliza como base al EBNA_{PC} [39]. Por último se desarrolla el algoritmo pEGNA_{EE}, que emplea en el aprendizaje el algoritmo EGNA_{EE} [73, 75]. El pEGNA_{EE} constituye un punto importante porque inicia los trabajos de paralelismo en EDA continuos. En el caso de todos los algoritmos desarrollados en [96] se evidencia el uso del modelo maestro/trabajador. Uno de los procesos que se comporta como maestro para el procesamiento de las secciones secuenciales, envía y recolecta información de los procesos trabajadores según las necesidades de cómputo. Una limitante de estos algoritmos, en ambientes distribuidos, es la necesidad de enviar la población para el aprendizaje del modelo probabilístico a todos los trabajadores. Esto provoca que para problemas complejos, con gran número de variables y poblaciones relativamente grandes, elevados volúmenes de información viajen por la red disminuyendo la escalabilidad del algoritmo.

Ahn y col. [2] proponen un esqueleto (framework) general para desarrollar algoritmos EDA distribuidos. A diferencia de los algoritmos distribuidos clásicos, donde la migración se desarrolla al intercambiar individuos entre las islas, este algoritmo simula el proceso al utilizar dos vectores de probabilidades; uno para mantener los individuos residentes en la isla (rPV) y otro para los que arriban a la isla (iPV). Con estos dos vectores se realiza todo el proceso, el cual consta de tres fases: una de generación, otra de selección y por último la fase de actualización (aprendizaje). El esqueleto tiene una gran ventaja al ser menor la cantidad de información a intercambiar entre los sub-algoritmos, lo que disminuye el tiempo global de comunicación del algoritmo. Este esquema funciona para algoritmos que asumen independencia entre las variables del problema y constituye una limitante en comparación a los algoritmos distribuidos que migran individuos [87].

Lobo y col. [79] presentan una arquitectura para el desarrollo de algoritmos masivamente paralelos basados en el cGA. La misma utiliza el modelo maestro/trabajador. El maestro se encarga de mantener

un vector de probabilidades para representar la distribución de los individuos. Cada trabajador ejecuta un algoritmo cGA y al final de cada generación se comunica con el maestro para actualizar la distribución. El esquema propuesto presenta tres ventajas: los bajos costos de sincronización, su tolerancia a fallos y su escalabilidad.

Mendiburu y col. [98] implementan una extensión al algoritmo pEBNA$_{\text{BIC}}$ que consiste en realizar la generación de nuevos individuos de forma distribuida. Cada trabajador recibe del maestro un orden de las variables y las probabilidades, para así generar una porción de la población y enviarla al maestro, al igual que su antecesor adolece de los altos costos de comunicación.

1.5.3. Medidas para el comportamiento paralelo

Para medir las ganancias en tiempo se utilizan fundamentalmente dos medidas, la ganancia en velocidad (speed-up) y la eficiencia. A continuación se definen ambos conceptos.

Ganancia en velocidad (speed-up)

El speed-up es una medida de la ganancia o incremento de la velocidad del algoritmo. Se define como el cociente entre el valor medio del tiempo de ejecución del algoritmo cuando se utiliza un procesador ($E[t_1]$), y el valor medio del tiempo de ejecución del algoritmo cuando se utilizan m procesadores ($E[t_m]$). La siguiente ecuación define el speed-up:

$$S_m = \frac{E[t_1]}{E[t_m]}$$

Tanto el speed-up como la eficiencia son sólo de utilidad para las implementaciones paralelas donde no varía el comportamiento del algoritmo. En el caso de las propuestas descentralizadas estas medidas se redefinen y se detallan en las secciones que se utilizan.

Eficiencia paralela

La eficiencia es una medida que se utiliza para normalizar el valor de speed-up a un determinado porciento. Se define como el cociente entre el speed-up (S_m) y el número de procesadores m, multiplicado por 100. La siguiente ecuación define la eficiencia.

$$e_m = \frac{S_m}{m} \cdot 100$$

1.6. Aplicaciones

Los EDA se aplican exitosamente en el entrenamiento de Redes Neuronales Artificiales (ANN) con varias propuestas interesantes. Para ajustar los pesos de una red neuronal, en el trabajo de Baluja [13] se utiliza el algoritmo PBIL y Zhang [169] lo logra en una ANN con estructura de árbol mediante algoritmos evolutivos Bayesianos. La red entrenada se emplea en la predicción de series de tiempo. Gallagher utiliza el algoritmo PBIL para visualizar la trayectoria del proceso de entrenamiento en la superficie del error [45].

Otros trabajos que manejan las técnicas evolutivas de estimación de distribuciones al entrenamiento de redes neuronales son los realizados por Galic y Maxwell [44, 93]. En ambos casos se aplica el algoritmo PBIL en el ajuste de los pesos de la red neuronal. En el primero la red es de tipo perceptron multicapa (MLP) y en el segundo es un modelo oculto de Markov (hidden Markov).

Cotta [32] en su estudio emplea los algoritmos UMDA_C y MIMIC_C al entrenamiento de redes neuronales. Se muestra la superioridad de estos métodos sobre el algoritmo genético clásico. Madera [88] aplica seis algoritmos EDA: tres discretos (UMDA, MIMIC_C, y BOA) y tres continuos (UMDA_C, MIMIC_C, y EGNA_{BGe}), al entrenamiento de una ANN del tipo MLP. El trabajo muestra la superioridad de los EDA de dominio continuo sobre los discretos en cuanto a la calidad de las soluciones encontradas así como en tiempo de ejecución.

En los trabajos de Endika [16, 15] se emplean los EDA en la solución del problema de la correspondencia de grafos, representándolos como un problema de optimización combinatoria con restricciones. Específicamente, el EBNA_{BIC} y varios algoritmos de dominio continuo ($\text{EMNA}_{\text{global}}$, EGNA_{BIC}, EGNA_{EE}, EGNA_{BGe}, UMDA_C).

1.7. Antecedentes de la investigación

A partir del objetivo general de la investigación de *crear esquemas paralelos de EDA con aprendizaje basado en pruebas de independencia y esquemas descentralizados para reducir el número de evaluaciones y el tiempo de ejecución* se pueden identificar los antecedentes en dos líneas de investigación fundamentales:

1. Aprendizaje de redes Bayesianas utilizando pruebas de independencia.
2. Utilización de los modelos paralelos y distribuidos en EDA.

Aprendizaje de redes Bayesianas utilizando pruebas de independencia

Los antecedentes de la investigación en este tema se remontan al estudio de los algoritmos de aprendizaje de redes Bayesianas en el campo del Aprendizaje Automatizado. En el trabajo de Neapolitan [108] se enumeran algunas de las ventajas que tienen los métodos de puntuación sobre los basados en restricciones. El autor expone que una estrategia interesante es la mezcla de ambos métodos, construyendo un esqueleto inicial de la red basado en pruebas de independencia y orientando dicho esqueleto con un método de puntuación. Este esquema se pone de manifiesto, con excelentes resultados, en varios trabajos de investigación [20, 163].

En el campo de los EDA siempre ha existido un "mito" en cuanto a la utilización de este tipo de método de aprendizaje. Los exponentes más relevantes utilizan métodos puros de optimización de métricas [39, 73, 128, 146, 102]. Son los EDA de estructuras simples los que utilizan fundamentalmente el aprendizaje basado en restricciones [34, 14, 132, 140, 150]. Un antecedente y motivación es el resultado reportado en los trabajos de Etxeberria, Larrañaga y col. [39, 73, 74], en ellos el peor comportamiento entre todos los métodos que aprenden redes generales es el del EBNA_{PC} (basado en pruebas de independencia) que utiliza como base el algoritmo de aprendizaje PC [154, 155, 156].

Estos antecedentes muestran que existen experiencias en el uso de los métodos de aprendizaje basados en restricciones, sin embargo, no existen trabajos previos a esta investigación sobre la utilización de modelos generales que aprenden la distribución de probabilidad conjunta de forma híbrida y el estudio de los mismos.

Utilización de los modelos paralelos en EDA

La utilización de técnicas paralelas en el entorno de los EDA recibe más atención que el tema anterior. Los trabajos se centran fundamentalmente en la utilización de modelos paralelos que permitan disminuir el tiempo de ejecución de los algoritmos para poder abordar problemas de mayores dimensiones. Producto a las mismas limitaciones en la investigación del tema anterior, no abundan las implementaciones paralelas para algoritmos que utilizan pruebas de independencia, a excepción del trabajo [96].

Un antecedente importante lo constituyen los trabajos [111, 112, 81] dedicados al aprendizaje de redes generales que utilizan optimización de métricas. En estos trabajos se evidencias algunas limitaciones en cuanto a la utilización de memoria compartida y a restricciones que se asignan al aprendizaje para mantener grafos dirigidos acíclicos. Mendiburu [96] por primera vez aplica técnicas paralelas híbridas, utilizando la programación multi-hilo y la programación distribuida mediante el intercambio de mensajes. Todos estos trabajos están encaminados a la reducción del tiempo de ejecución del algoritmo EDA.

Tanto las técnicas de programación paralela utilizadas, como los algoritmos de aprendizaje paralelizados hacen que los EDA paralelos y/o descentralizados de esta tesis se diferencien de los anteriormente mencionados y será objeto de estudio en los capítulos 3 y 4.

1.8. Conclusiones

En este capítulo se introdujo una familia de algoritmos que pertenecen al campo de la computación evolutiva, los Algoritmos de Estimación de Distribuciones. En los últimos años se han propuesto varios algoritmos EDA para dominio discreto y continuo. El objetivo del capítulo es introducir el esquema general de EDA, su relación con los modelos gráficos probabilísticos, explicar las diferentes propuestas secuenciales y paralelas, así como aplicaciones prácticas resueltas con EDA.

De esta manera, la tendencia actual en el desarrollo de algoritmos EDA se enfoca hacia la utilización de los esquemas basados en optimización de métricas y su paralelización. A partir de esta razón, es importante enmarcar el problema de investigación en la necesidad de disponer de algoritmos EDA, que combinen la disminución del tiempo de ejecución con la disminución del número de evaluaciones.

CAPÍTULO 2

ALGORITMOS DE ESTIMACIÓN DE DISTRIBUCIONES BASADOS EN PRUEBAS DE INDEPENDENCIA. MODELOS BAYESIANOS SECUENCIALES

En este capítulo se investigan tres tipos de algoritmos EDA que tienen en común la utilización de pruebas de independencia durante el aprendizaje de modelos Bayesianos de sus distribuciones de búsqueda[1]. En la literatura especializada se utiliza el término ¨algoritmo basado en restricciones¨ para referirse a la existencia de procedimientos de búsqueda de relaciones –restricciones– de independencia entre las variables del problema. Con esta idea los algoritmos de este capítulo son referidos como *Algoritmos de Estimación de Distribuciones Basados en Restricciones* (CBEDA).

Los modelos secuenciales no son el destino final de esta exploración científica, pero si constituyen la columna vertebral del trabajo. Si se quiere construir modelos paralelos EDA de estado del arte es ineludible la utilización de modelos secuenciales de la misma categoría como punto de partida. Desafortunadamente, las propuestas algorítmicas existentes al comenzar esta investigación no saciaron las inquietudes científicas e hipótesis iniciales de trabajo. Por ejemplo, la conveniencia de utilizar esquemas basados en pruebas de independencia como componente esencial de los EDA paralelos. Esta es la razón de la inclusión de este capítulo en la tesis, exponer resultados y propuestas sobre modelos secuenciales.

2.1. Introducción

Los avances científicos en los últimos años y los retos actuales, por ejemplo en el campo de la Bioinformática, hacen del aprendizaje de redes Bayesianas de altas dimensiones, una tarea de altísima prioridad e importancia. En el contexto de los EDA esto equivale a problemas de optimización con cientos, miles o decenas de miles de variables. Sin embargo, los algoritmos de aprendizaje existentes y los EDA que los utilizan, no escalan apropiadamente cuando aumenta la dimensión del problema, y como consecuencia falla la optimización. Es obvio que se necesitan nuevas propuestas algorítmicas que aceleren el proceso de aprendizaje sin comprometer la calidad de la red aprendida. Los algoritmos que se proponen en este trabajo constituyen avances importantes en esta dirección.

Para mejorar los EDA, particularmente los que se reportan en este capítulo, el aprendizaje constituye el principal objeto de atención. El aprendizaje de una red Bayesiana a partir de datos es un problema NP-Duro [27, 28]. Vale notar el comentario de Silverstein [149] que expresó que: "*la inferencia del modelo causal completo en aplicaciones que utilizan conjuntos de datos de altas dimensiones es*

[1]Las distribuciones de los conjuntos seleccionados en cada iteración del algoritmo evolutivo.

esencialmente imposible". Ideas como estas, conducen a una reflexión importante con relación a la manera- de abordar estos problemas:

1. *Utilización de modelos gráficos sencillos*. La idea de los modelos de complejidad acotada ha sido estudiada por Ochoa y colaboradores [151, 152, 115, 150]. El primero de los algoritmos que se estudia en el capítulo (sección 2.3) pertenece a este enfoque y trabaja con modelos Bayesianos simplemente conectados. Debido a su bajo costo, este tipo de algoritmo puede ser una correcta elección en algunos problemas de altas dimensiones, a pesar de su limitada capacidad para codificar las relaciones existentes en el problema. Más aún, se puede mostrar la existencia de clases de problemas, como las B-funciones poliárboles [118, 120], donde el desempeño de estos algoritmos es difícil de superar. Existen también ejemplos de aplicaciones reales donde las estructuras simples han sido exitosas [141].
2. *Mejorar y/o resumir los modelos más complejos*. Esta vía conduce a la utilización de modelos más generales pero con esquemas de aprendizaje más eficientes y eficaces, como los que serán objeto de estudio en las secciones 2.4 y 2.5.

Nótese que en ambos casos el uso de técnicas paralelas ofrece adicionalmente escalabilidad en el tiempo.

Los tres algoritmos que se estudian en este capítulo utilizan esquemas híbridos de aprendizaje. Inicialmente se aprende un esqueleto de la red Bayesiana utilizando pruebas de independencia y posteriormente se orienta el esqueleto mediante un algoritmo de puntuación. Esta técnica, que se utilizó por primera vez en el contexto de los EDA en el algoritmo PADA [115, 151], resultó excepcionalmente útil en el aprendizaje de redes generales [163].

Las razones que motivaron a investigar el tema tratado en este capítulo se pueden resumir de la siguiente manera:

- La necesidad inmediata de disponer en EDA, de algoritmos capaces de abordar problemas de optimización de mayor complejidad reduciendo el número de evaluaciones.
- La necesidad de disponer de un estudio que analice las bondades de los algoritmos de aprendizaje de redes Bayesianas basados en restricciones en la construcción de la distribución de búsqueda del EDA.

Durante el desarrollo del capítulo se pretende conseguir los siguientes objetivos:

1. A partir de recientes resultados en el área de aprendizaje automático, evaluar de forma crítica el estado actual de la utilización de los algoritmos de aprendizaje basados en restricciones en el contexto de los optimizadores Bayesianos.
   a) Evaluar la conveniencia de la generalización del esquema de aprendizaje híbrido utilizado en PADA [150] a nuevos algoritmos EDA multi-conectados.
   b) Comparar críticamente los algoritmos de aprendizaje TPDA [26] y MMHC [20, 163].

2. Presentar nuevas propuestas EDA, basados en algoritmos de aprendizaje híbrido de redes Bayesianas, a partir de las lecciones obtenidas del objetivo anterior y diseñados para dominio discreto y continuo.
3. Diseñar e implementar en *C++* los algoritmos propuestos.
4. Evaluar empíricamente el desempeño de los nuevos EDA, comparándolos con otros algoritmos del estado del arte y utilizando las B-funciones para el estudio en dominio discreto.

Para cumplimentar los objetivos planteados, se comienza por un análisis de los métodos de aprendizaje Bayesianos basados en restricciones, comparándolos con los métodos de puntuación. El análisis se divide en dos partes: la primera dedicada al papel de los algoritmos basados en restricciones dentro del campo del aprendizaje de máquina y la segunda dedicada a los EDA que utilizan este tipo de método.

La hipótesis fundamental de este capítulo es que los algoritmos basados en restricciones deben jugar un papel importante en la práctica de los EDA, aunque seguramente combinados con esquemas de puntuación. Esta hipótesis se sustenta en los resultados obtenidos por el grupo de investigación del autor con respecto al uso de esquemas híbridos para el aprendizaje de estructuras simplemente conectadas [152, 150, 84, 85]. Por este motivo, se prosigue con un análisis de esta clase de algoritmos para luego intentar generalizar las ideas a modelos multi-conectados.

En un segundo momento de la investigación se introducen dos nuevos esquemas EDA donde el aprendizaje de la distribución de búsqueda se realiza mediante métodos basados en restricciones. Se ha propuesto llamarles CBEDA a las propuestas algorítmicas de este capítulo. Posteriormente, se extiende el estudio a dominio continuo con la propuesta de un nuevo algoritmo.

Por último, se presentan un conjunto de resultados experimentales que persiguen dar una visión general del comportamiento de los CBEDA. Se incluyen comparaciones con otros EDA que utilizan métricas de puntuación y que se encuentran en el estado del arte. Para dominio continuo también se comparan con EDA basados en cópulas.

2.2. Pruebas de independencias en EDA

En esta sección se discuten los principales métodos de aprendizaje de redes basados en pruebas de independencia, orientado a su papel en la optimización mediante EDA. En particular, es de interés contestar a la siguiente pregunta: ¿qué método es superior, el de aprendizaje basado en métricas de puntuación o el basado en restricciones? No se ha encontrado una respuesta definitiva a esta pregunta en la literatura consultada. Mucho menos, si lo que interesa son las bondades de estos métodos con respecto a los EDA.

En el libro publicado sobre el tema del aprendizaje de redes Bayesianas [108], Neapolitan compara los métodos de aprendizaje por puntuación con los de detección de independencias, enfatizando en tres ventajas fundamentales de los primeros sobre los basados en restricciones:

1. Evitan errores sobre decisiones categóricas con relación al modelo que se aprende cuando existe poca disponibilidad de información.

2. Manipulan instancias con valores sin especificar en sus rasgos. Normalmente los métodos basados en restricciones no toman en cuenta estas instancias.
3. Detectan modelos que no son inferidos por los métodos basados en restricciones.

Neapolitan se basó en un estudio similar desarrollado con anterioridad por Heckerman y colaboradores [62], trabajo recomendado a los lectores interesados en el tema. Como resultado del estudio, Neapolitan afirmó que los métodos basados en restricciones abordan problemas con variables ocultas cuando los métodos de puntuación no descubren dichas variables. Sobre la base de este argumento Neapolitan propuso un esquema híbrido para el aprendizaje de la red. Como se verá a lo largo de este capítulo, a esta misma conclusión llegaron varios autores de manera independiente. Por ejemplo, esta es la estrategia que se sigue en los algoritmos PADA [150], MMHC [20, 163] y en la versión del TPDA [26] propuesta en este trabajo (sección 2.4).

A continuación se hace una breve presentación de los esquemas básicos de aprendizaje de redes, seguida de una exposición de los EDA que utilizan estos métodos.

2.2.1. Métodos básicos de aprendizaje de redes Bayesianas

Uno de los primeros algoritmos propuestos para el aprendizaje de redes Bayesianas que utilizan pruebas de independencia es el conocido algoritmo Chow-Liu [29]. El mismo construye el árbol de cubrimiento de peso máximo (MWST) de la matriz de información mutua del problema y produce la estructura correcta si los datos siguen una distribución de probabilidad con forma de árbol. De lo contrario, se aprende la mejor aproximación con esta forma. Las posiciones de la matriz con menor valor que un umbral representan relaciones de independencia. Por esta razón su construcción constituye un proceso de detección de dependencias. No obstante, el algoritmo sigue también el esquema general de los métodos de optimización de métricas, pues su operación maximiza una cierta medida de calidad de la red.

Una extensión del algoritmo Chow-Liu y la primera propuesta para el aprendizaje de poliárboles es introducida por Rebane y Pearl [133]. Este algoritmo permite representar relaciones de dependencia de orden mayor que su antecesor, mientras preserva muchas de sus ventajas. Al igual que la propuesta de Chow y Liu, este método utiliza un esquema híbrido, pero en este caso se agrega una fase de orientación del esqueleto utilizando pruebas de dependencias entre pares de variables.

Otros dos métodos que utilizan pruebas de independencia para el aprendizaje de poliárboles son las propuestas de Acid y De Campos [1, 35]: una exacta y otra aproximada (PA). La idea principal del algoritmo PA es conservar las aristas con mayor valor de una medida de dependencia global (definición 7). El algoritmo construye la matriz de información mutua marginal y la matriz de dependencia global. La primera se utiliza para crear el esqueleto, mientras que ambas intervienen en el proceso de orientación de un subconjunto de las aristas del problema. A diferencia de las anteriores propuestas la complejidad de este algoritmo es cúbica ($O(n^3)$).

Uno de los métodos más conocidos entre los que utilizan pruebas de independencia para el aprendizaje de redes multi-conectadas es el algoritmo PC [154, 156]. El algoritmo parte de una red totalmente

conectada y elimina aristas entre nodos que pueden ser separados dado algún subconjunto de sus variables adyacentes. Tiene una naturaleza ávida al tomar la decisión de eliminar una arista y esta no se considera más durante las siguientes iteraciones. Este algoritmo tiene algunas limitaciones como son su complejidad exponencial y su naturaleza ávida lo que limita el análisis de estructuras alternativas.

2.2.2. Detección de independencias en EDA

En esta sección se presentan un conjunto de EDA que utilizan en su funcionamiento la detección de independencias, específicamente, combinaciones de algunas de las técnicas vistas en la sección anterior. Todos los algoritmos a partir del conjunto (población) seleccionado, en el instante (generación) de tiempo t, aprenden el modelo probabilístico que posteriormente se muestrea para obtener la población de la generación $t+1$.

Dependencias bivariadas

Dentro del grupo de EDA que utilizan dependencias bivariadas (consideran solo dependencias marginales) se encuentra el algoritmo MIMIC (sección 1.4.2). También se incluyen algunas modificaciones a este algoritmo para la solución de problemas de satisfacción con restricciones [56] y un operador de mutación para mejorar las cualidades del algoritmo [57].

Una extensión del algoritmo MIMIC lo constituye la propuesta de De Bonet: Combinando Optimizadores Locales con Árboles de Información Mutua (COMIT) [14]. Este algoritmo representa la distribución de probabilidad mediante una red Bayesiana con estructura de árbol lo que constituye una extensión a las cadenas que forma el algoritmo MIMIC, y por ende, es capaz de representar un mayor número de distribuciones de probabilidad. Para encontrar el árbol que mejor se ajusta a los datos, COMIT utiliza el algoritmo MWST de Chow-Liu [29].

Otro de los exponentes de la familia de algoritmos que utilizan dependencias de segundo orden es el BMDA [132]. El algoritmo utiliza los árboles de dependencia acíclicos pero con la particularidad de que no existe una sola componente, o sea, construye bosques de árboles de dependencia. Para determinar las dependencias bivariadas se utiliza la prueba χ^2 de Pearson [40]. Este algoritmo es capaz de optimizar funciones cuya dependencia entre las variables es de hasta segundo orden pero falla ante problemas con dependencias de orden superior.

El algoritmo Evolutivo con Distribución Factorizada basado en Árboles (Tree-FDA) [140] utiliza como modelo probabilístico a los árboles. A diferencia del BMDA el núcleo del Tree-FDA es el algoritmo Chow-Liu para la construcción de árboles de cubrimiento de peso máximo. De forma similar Soto [150] propone un algoritmo EDA que utiliza los árboles para representar el modelo probabilístico del conjunto seleccionado.

Dentro de los algoritmos que utilizan detección de independencias de orden cero (dependencias marginales) el que mayor expresividad tiene es Algoritmo con Distribución Factorizada Basado en Poliárboles (PADA poliárbol - caso cuadrático, PADA_{p2}). Este algoritmo representa la distribución

de probabilidad mediante una estructura de poliárbol y se inspira en el aprendizaje de poliárboles propuesto por Rebane y Pearl [133].

Dependencias trivariadas

Dentro de los algoritmos EDA que utilizan pruebas de detección de independencias de hasta tercer orden se encuentra el $PADA_{t3}$ [150]. El mismo basa su funcionamiento en el algoritmo para Aprendizaje de Árboles Aproximado (TA) propuesto en [35]. Evidentemente, el algoritmo TA tiene mayor complejidad que Chow-Liu pues realiza pruebas de independencia condicional de primer orden. La idea de este algoritmo es preservar los arcos de mayor peso utilizando la medida de dependencia global (definición 7).

De forma similar a $PADA_{t3}$, el $PADA_{p3}$ [151] basa su funcionamiento en pruebas de independencia y utiliza la medida de dependencia global para preservar las aristas de mayor peso. A diferencia de $PADA_{t3}$, la estructura construida por este algoritmo tiene forma de poliárbol por lo que tiene mayor expresividad. $PADA_{p3}$ utiliza una versión modificada del algoritmo PA [35] llamada LPA. Por primera vez se utiliza un esquema híbrido de métodos de detección de independencias y métodos de puntuación para mejor la calidad de la red aprendida por el EDA. Este algoritmo fue posteriormente mejorado [115, 150] para permitir al modelo representar la clase general de redes Bayesianas simplemente conectadas. También el algoritmo de muestreo de PADA se modifica [118] para utilizar marginales de hasta segundo orden correlacionado con el concepto de máxima entropía, implicando de forma inmediata una aceleración de la convergencia del algoritmo a menor costo. En la sección 2.3 se presenta con más detalles el algoritmo PADA.

Dependencias generales

La mayoría de los algoritmos EDA que son capaces de aprender redes Bayesianas generales utilizan los métodos de puntuación [39, 74, 128]. Sin embargo, uno de los algoritmos propuestos en los trabajos [39, 74] utiliza un algoritmo de aprendizaje basado en restricciones. Los autores llamaron a este algoritmo Algoritmo Evolutivo con Estimación de Redes Bayesianas ($EBNA_{PC}$). El subíndice PC indica que se utiliza el algoritmo PC [154, 156] para el aprendizaje de la red Bayesiana. El $EBNA_{PC}$ es superado en la eficiencia evaluativa por los algoritmos$EBNA_{BIC}$ y $EBNA_{K2+pen}$ [39, 74].

2.2.3. Pruebas de independencia en EDA Markovianos

Otra de las construcciones matemáticas que utilizan los EDA para representar el modelo probabilístico del conjunto seleccionado son las redes de Markov. La principal característica de este tipo de red es que el modelo gráfico se representa por un grafo no dirigido. Por ejemplo, una estructura de árbol es una red de Markov y si se selecciona un nodo raíz inmediatamente se infiere la dirección de las aristas, convirtiéndose en una red Bayesiana.

Dentro de las alternativas más promisorias para tratar problemas de optimización, que presentan interacciones entre las variables, se encuentra la combinación de modelos probabilísticos (mezclas). Las primeras investigaciones en EDA discretos que utilizan mezclas de distribuciones se presentan

en los trabajos de Santana y col. [144, 145]. En ellos se emplean las mezclas de árboles como alternativa para el aprendizaje del modelo probabilístico del algoritmo EDA, específicamente el MT-FDA. Santana [138] extiende los resultados de otros trabajos [144, 145] y propone un algoritmo EDA basado en mezclas de distribuciones con aproximaciones de Kikuchi [167, 168].

Ochoa y col. [121] proponen el FDA-learning (sección 1.4.2). La importancia de este tipo de estructuras es que cumplen con la propiedad de intercepción corrida [77] que permite, a partir del grafo cordal, construir un árbol de cliques. Esto es muy importante para algoritmos que aprenden frecuentemente el modelo probabilístico a partir de los datos, como es el caso de los algoritmos EDA.

En los trabajos de Santana [138, 137] se propone la utilización de las aproximaciones de Kikuchi [167, 168] para la construcción de factorizaciones inválidas, el algoritmo resultante se nombra MN-EDA. Este algoritmo es una extensión del MN-FDA [136, 138] que utiliza factorizaciones válidas al igual que el FDA-learning. La base de estos algoritmo es el algoritmo de aprendizaje basado en restricciones PC [154, 155, 156] con la restricción que solo se hacen pruebas de independencia de orden cero, uno y dos.

Gámez y col. [46, 47] proponen la utilización de redes de dependencias para proponer un nuevo EDA llamado EDNA (sección 1.4.2). El algoritmo considera solo dependencias bivariadas para estimar la distribución de búsqueda del conjunto seleccionado.

2.2.4. Complejidad de las pruebas de independencia

De forma general, el costo computacional para computar $Ind(X_i, X_j \mid X_k)$ es exponencial en el tamaño del conjunto X_k. En otras palabras, se requiere un tiempo proporcional al producto de las cardinalidades de X_i, X_j y el conjunto X_k. Por otro lado, el tiempo de cómputo es lineal al tamaño del conjunto de datos.

En el cómputo de las dependencias entre las variables es donde mayor esfuerzo se puede realizar para lograr implementaciones paralelas eficientes. La idea es dedicar un conjunto de procesos a estos cálculos para acelerarlos.

2.3. Algoritmos simplemente conectados

Esta sección trata sobre PADA, un EDA basado en el aprendizaje de poliárboles, por lo que generaliza a todos los modelos simplemente conectados: árboles, cadenas, bosques, independencia.

Antes de mostrar el funcionamiento del algoritmo se necesita definir los conceptos de *dependencia marginal*, *dependencia condicional* y *dependencia global.*

Definición 5. *La medida de la* información mutua marginal $Ind(X_i, X_j) = Dep(X_i, X_j)$ *de las variables aleatorias* X_i *y* X_j *se define como:*

$$Ind(X_i, X_j) = \sum_{x_i, x_j} p(x_i, x_j) \cdot \log \frac{p(x_i, x_j)}{p(x_i) \cdot (x_j)}$$

Definición 6. *La medida de la* información mutua condicional $I(X_i,X_j|Z) = Dep(X_i,X_j|Z)$ *de las variables aleatorias* X_i *y* X_j *dado el conjunto* Z *se define como:*

$$I(X_i,X_j|Z) = \sum_{x_i,x_j,z} p(x_i,x_j,z) \cdot \log \frac{p(x_i,x_j,z) \cdot p(z)}{p(x_i,z) \cdot p(x_j,z)}$$

Definición 7. *La medida de la* dependencia global $Dep_g(X_i,X_j)$ *de las variables aleatorias* X_i *y* X_j *dada* X_k *se define como:*

$$Dep_g(X_i,X_j) = \min_{\{X_k\}}(Dep(X_i,X_j), I(X_i,X_j|X_k))$$

Para la estimación de la distribución de probabilidad, se utilizan diferentes algoritmos de aprendizaje, Soto [150] estudia cinco algoritmos para el aprendizaje de poliárboles. Todos utilizan pruebas de dependencias para construir el esqueleto del poliárbol. De forma general todas las propuestas siguen el mismo esquema: parten de un grafo totalmente desconectado e incrementan el tamaño de la red adicionando los arcos de mayor peso, según dos métricas definidas para los arcos $\langle X_i,X_j \rangle$: la dependencia marginal $Dep(X_i,X_j)$ y la dependencia global (definiciones 5, y 7). Estas ecuaciones permiten obtener el grado de dependencia que existe entre dos variables.

En el algoritmo 2.1 se muestra el pseudo-código del *LPA* [151] que toma como punto de partida al método PA desarrollado por Acid y de Campos [1]. El número máximo de arcos que se insertan es $n-1$, restricción que garantiza que la estructura aprendida es simplemente conectada. Este algoritmo difiere del PA en dos cuestiones fundamentales, primero utiliza dos umbrales para la determinación de la dependencia marginal y la condicional $(\varepsilon_0, \varepsilon_1)$, introduce modificaciones en la forma de orientar las aristas.

El algoritmo LPA comienza con un grafo sin aristas G y una lista vacía L para almacenar las relaciones de dependencias detectadas en los datos (línea 2). Para cada par de variables $\langle X_i,X_j \rangle$, se computa la dependencia marginal (definición 5) y los arcos con $Dep(X_i,X_j) > \varepsilon_0$ son almacenados en la lista L (líneas 3 - 8). Con los arcos resultantes en L, el *LPA* computa la información mutua condicional (definición 6), eliminando los arcos con $Dep(X_i,X_j|X_k) < \varepsilon_1$(líneas 9 - 17). Para todo los arcos $\langle X_i,X_j \rangle$ en L, se calcula el valor de $Dep_g(X_i,X_j)$ (línea 18 - 20), utilizando estos valores para ordenar la lista L (línea 21). A continuación se construye el esqueleto del poliárbol (línea 22 - 27). Como se observa, los arcos se insertan en el grafo G recorriendo la lista L, teniendo en cuenta que al insertar la arista $\langle X_i,X_j \rangle$ no se crea un ciclo no dirigido en el grafo resultante. Con el esqueleto construido, el algoritmo comienza a orientar las aristas en G (líneas 28 - 32). Para esto se analiza cada sub-grafo $X_i - X_k - X_j$ y si $Dep(X_i,X_j|X_k) > Dep(X_i,X_j)$ se crea un patrón cabeza-cabeza: $X_i \rightarrow X_k \leftarrow X_j$. El resto de los arcos se orientan como se explica a continuación.

Algoritmo 2.1 Algoritmo para el aprendizaje de poliárboles

procedimiento LPA(*conjunto de datos D*, ε_0, ε_1)
    Comenzar con un grafo sin aristas G y una lista vacía L
    para cada $X_i, X_j \in X \mid i \neq j$ **hacer**
        Computar $Dep(X_i, X_j)$
        si $Dep(X_i, X_j) > \varepsilon_0$ **entonces**
            Adicionar el arco $\langle X_i, X_j \rangle$ a L
        fin si
    fin para
    para cada $\langle X_i, X_j \rangle$ en L **hacer**
        para cada $X_k \in \mathbf{X}$ **hacer**
            Computar $Dep(X_i, X_j \mid X_k)$
            si $Dep(X_i, X_j) < \varepsilon_1$ **entonces**
                Eliminar el arco $\langle X_i, X_j \rangle$ de L
                Seleccionar el próximo arco $\langle X_i, X_j \rangle$ en L
            fin si
        fin para
    fin para
    para cada $\langle X_i, X_j \rangle$ en L **hacer**
        Computar $Dep_g(X_i, X_j)$
    fin para
    Ordenar L en orden decreciente del valor de $Dep_g(X_i, X_j)$
    repetir
        Seleccionar la próxima arista $\langle X_i, X_j \rangle$ en L
        si $\langle X_i, X_j \rangle$ no crea un ciclo en G **entonces**
            Adicionar $\langle X_i, X_j \rangle$ a G
        fin si
    hasta $n-1$ aristas hayan sido adicionadas a G
    para cada $X_i - X_k - X_j \in G$ **hacer**
        si $Dep(X_i, X_j | X_k) > Dep(X_i, X_j)$ **entonces**
            Orientar patrón cabeza-cabeza $X_i \rightarrow X_k \leftarrow X_j$
        fin si
    fin para
    Orientar el resto de las aristas aplicando alguna función de costo
    Computar $p(x) = \prod_{i=1}^{n} p(x_{j1(i)}, \ldots, x_{jr(i)})$
fin procedimiento

2.3.1. Orientación de las aristas en el LPA

De forma general, los algoritmos basados en restricciones utilizan los resultados obtenidos en las pruebas de independencia ejecutadas durante el proceso de construcción del esqueleto para orientar algunas de las aristas. El tipo de arista que se orienta es aquella que forman los patrones *cabeza-cabeza*.

La principal desventaja que presenta este método es que no se orienta totalmente el esqueleto, siempre quedan aristas a las cuales les aplica otra técnica para inferir su dirección. Generalmente, se emplea un proceso de dirección aleatoria y se minimiza la creación de nuevos patrones *cabeza-cabeza.* Una segunda desventaja es que durante el proceso de orientación surgen distintos conflictos [152, 150]. Estos están relacionados con la identificación de los patrones *cabeza-cabeza*. En este caso se identifican dos tipos de problemas:

- En el grafo están representados patrones cabeza-cabeza que no lo son
- En el grafo no están representados patrones cabeza-cabeza que sí lo son

Para reducir los efectos negativos de los conflictos, los autores proponen una modificación importante al algoritmo [152, 150] y es incorporar un procedimiento ávido que optimiza la métrica BIC en el proceso de orientación de las aristas. La importancia de estos resultados se refleja en la cita de Larrañaga y Lozano [71] al referirse al LPA:

"In Soto et al. (1999) the factorization is done by means of a Bayesian network with a polytree structure (no more than one undirected path connecting every pair of variables). The proposed algorithm is called PADA (Polytree Approximation of Distribution Algorithms) and can be considered a hybrid between a method proposed by Acid and de Campos (1995) for detecting independencies with a procedure based on score and search".

En resumen, se afirma que el LPA está formado por tres fases: 1) creación del esqueleto; 2) orientación de las estructuras cabeza-cabeza no conflictivas y 3) orientación del resto de las aristas por medio de la BIC.

Estos argumentos justifican una de las decisiones tomadas en la implementación del algoritmo TPDA, en el cual se sustituyó el esquema de orientación propuesto por Cheng y col. [26] por otro basado en optimización de métricas.

Estos resultados demuestran lo acertado de la decisión de profundizar en el estudio de las técnicas basadas en restricciones y específicamente en los métodos híbridos. Todo esto se refleja en las nuevas propuestas algorítmicas que se presentan en esta tesis y cuya validez queda demostrada en los resultados experimentales.

2.4. Del LPA a la mejora del TPDA

A partir de los resultados obtenidos en el estudio de las estructuras simplemente conectadas en EDA se estudian nuevos algoritmos de aprendizaje que se basan en pruebas de independencia. No solo estudiarlos, sino proponer nuevas modificaciones para lograr un algoritmo del estado del arte, robusto y con excelentes cualidades para su posterior paralelización.

2.4.1. El algoritmo de Análisis de Dependencias de Tres Fases (TPDA)

El TPDA es un procedimiento de tres etapas que utiliza los métodos basados en restricciones para el aprendizaje de redes Bayesianas. En la primera etapa, el algoritmo construye un acercamiento inicial (*borrador*[2]) de la red Bayesiana. El TPDA utiliza el procedimiento Chow-Liu [29] para la construcción de la aproximación inicial de la red Bayesiana que se aprende. En la segunda etapa el algoritmo realiza un *aumento* o *engrosado*[3] de la red resultante del paso anterior. Se adicionan arcos cuyos nodos no se separan utilizando un conjunto de pruebas de independencia. Al finalizar esta etapa la red que se aprende contiene todos los arcos del modelo de independencia e incluso más. En la tercera etapa se analizan todos los arcos y se eliminan aquellos que no pertenecen al modelo (*refinado*[4] de la red), utilizando pruebas de independencia condicional. Esta etapa es muy similar a la que realiza el MMHC (sección 2.5) cuando elimina los falsos positivos. Por último se ejecuta un procedimiento para orientar los arcos esenciales y producir un *grafo esencial*[5] [156]. La red Bayesiana resultante tiene arcos sin orientar cuya información no es relevante.

Evidentemente, para poder utilizar el algoritmo en el entorno de los EDA se necesita que todos los arcos del grafo tengan dirección para lo cual en esta tesis se proponen cambios al proceso de orientación de aristas. Ocenasek [111, 110] propone el algoritmo pBOA, una implementación paralela de BOA [128]. Para lograr mantener el modelo gráfico acíclico y con un mejor comportamiento paralelo, el autor genera una permutación aleatoria en cada paso de la evolución del algoritmo. De esta forma se restringe el espacio de búsqueda de la red Bayesiana que mejor se ajusta a los datos. Posteriormente Pelikan y Laury [130] estudian la influencia de dicha permutación en la escalabilidad del algoritmo hBOA. Los autores muestran que no se deterioran los resultados a pesar de reducirse el espacio de posibles soluciones. En este trabajo se conjetura lo planteado anteriormente. En el caso de la optimización varía el comportamiento del algoritmo cuando el modelo probabilístico difiere en algunos arcos del modelo real de la función objetivo. Ochoa y Soto [118] muestran cómo la diferencia de solo un arco en el modelo, afecta los resultados del algoritmo de optimización.

A continuación se presenta una breve panorámica del TPDA; se refiere al lector a los trabajos de Cheng y col. [26] y a sus referencias para un detallado tratamiento del tópico. Se analizan las etapas dos y tres, la primera se ha estudiado con profundidad en varios trabajos [29, 151, 115, 150] y constituye la base de otros algoritmos de aprendizaje como el LPA (sección 2.3).

El algoritmo 2.2 muestra el pseudo-código original propuesto para el TPDA [26]. Comienza con un grafo sin aristas G y una lista vacía L para almacenar las relaciones dependencias entre las variables detectadas en los datos (línea 3). Para cada par de variables $\langle X_i, X_j \rangle$ el algoritmo computa la dependencia marginal (definición 5) y los arcos con $Dep(X_i, X_j) > \varepsilon$ se almacenan en la lista L (líneas 4

[2] *drafting*
[3] *thickening*
[4] *thinning*
[5] Tiene los mismos arcos que la red Bayesiana resultante y las estructuras *cabeza-cabeza*

Algoritmo 2.2 Algoritmo de Análisis de Dependencias de Tres Fases

procedimiento TPDA(*conjunto de datos D, umbral* ε) ▷ retorna la red Bayesiana BN
    Sea $X = \{atributos\ en\ D\}$
    Comenzar con un grafo sin aristas G y una lista vacía L
// Borrador
    para cada $X_i, X_j \in X \mid i \neq j$ **hacer**
        si $Dep(X_i, X_j)$ **entonces**
            Adicionar el arco $\langle X_i, X_j \rangle$ a L
        fin si
    fin para
    Ordenar L en orden decreciente del valor de $Dep(X_i, X_j)$
    repetir
        Seleccionar la próxima arista $\langle X_i, X_j \rangle$ en L
        si $\langle X_i, X_j \rangle$ no crea un ciclo en G **entonces**
            Adicionar $\langle X_i, X_j \rangle$ a G
            Eliminar $\langle X_i, X_j \rangle$ de L
        fin si
    hasta $n-1$ aristas hayan sido adicionadas a G
// Engrosado
    para cada $\langle X_i, X_j \rangle \in L$ **hacer**
        si $EdgeNeeded\_H(G, X_i, X_j, D, \varepsilon)$ **entonces**
            Adicionar $\langle X_i, X_j \rangle$ a G
        fin si
    fin para
// Refinado
    para cada $\langle X_i, X_j \rangle \in G$ **hacer**
        si existe otro camino conectando a X_i con X_j **entonces**
            $G' = G - \langle X_i, X_j \rangle$
            si $\neg EdgeNeeded\_H(G, X_i, X_j, D, \varepsilon)$ **entonces**
                $G = G'$
            fin si
        fin si
    fin para
    para cada $\langle X_i, X_j \rangle \in G$ **hacer**
        si X_i tiene otros tres vecinos diferentes de X_j ó X_j tiene otros tres vecinos distintos de X_i **entonces**
            $G' = G - \langle X_i, X_j \rangle$
            si $\neg EdgeNeeded(G, X_i, X_j, D, \varepsilon)$ **entonces**
                $G = G'$
            fin si
        fin si
    fin para
    retornar $OrientEdges(G, D)$
fin procedimiento

- 8). La lista L se ordena de forma descendente según el valor de $Dep(X_i, X_j)$ (línea 9). A continuación se construye el esqueleto inicial de la red Bayesiana (líneas 10 - 16). Como se observa, los arcos se insertan en el grafo G recorriendo la lista L, teniendo en cuenta que al insertar la arista $\langle X_i, X_j \rangle$ no se crea un ciclo no dirigido en el grafo resultante. Posteriormente comienzan las dos fases principales del algoritmo, primero el esqueleto inicial se aumenta comprobando la necesidad de adicionar nuevos arcos (líneas 17 - 21). Se comprueba el concepto de d-separación [123, 124] utilizando la definición de independencia mutua condicional (definición 6). Se utiliza como conjunto condicionante los nodos que se encuentran en los caminos de adyacencia entre los dos nodos examinados y cualquier subconjunto que se pueda formar a partir de ellos. Posteriormente, comienza el refinado de la red, eliminando todos los arcos que se adicionan erróneamente en las dos primeras fases (líneas 22 - 37). El algoritmo comprueba para cada arco del grafo si no es d-separado por un subconjunto de nodos, en tal caso se mantiene el arco en el grafo. Es importante notar que tanto en la fase dos como en la tres se utiliza una heurística para la comprobación de la necesidad de mantener un arco (EdegeNeede_H). Alternativamente, hay un conjunto de arcos para los cuales es necesario chequear su permanencia en el grafo mediante un procedimiento exacto (EdgeNeeded). Con el esqueleto construido, el algoritmo orienta las aristas en G (línea 38).

PROBLEMA 1: El valor umbral ε, basado en la teoría de la información, es difícil de ajustar y depende directamente del tamaño de la muestra utilizada para el aprendizaje [150].

En la propuesta que se presenta en esta tesis se cambia el procedimiento para orientar los arcos por uno que utiliza un método de puntuación, similar al que se utiliza en los algoritmos LPA y MMHC. Esta modificación es uno de los aportes principales a la propuesta inicial del algoritmo TPDA (sección 2.4.4).

2.4.2. Comprobación de la necesidad de una arista

El método que utiliza el TPDA para determinar si una arista $\langle X, Y \rangle$ pertenece al grafo es cuantitativo [26] – midiendo la cantidad de flujo de información entre los nodos X e Y, para un conjunto condicionante C. La rutina $EdgeNeeded\_H$ (algoritmo 2.3) comienza con un conjunto C que garantiza ser un superconjunto del conjunto de corte ideal. Este entonces trata de identificar y eliminar , uno por uno, los nodos inadecuados de C realizando pruebas de información mutua. Este proceso requiere solo $O(k^2)$ pruebas de independencia, donde k es la cardinalidad máxima de cualquier conjunto de corte. El lector interesado en la demostración de la validez del procedimiento $EdgeNeeded\_H$ se remite a [26].

El procedimiento EdgeNeeded_H utiliza la heurística que al eliminar un padre de X (o Y) del conjunto condicionante raramente cierra cualquier camino entre X e Y. Sin embargo, esto sucede para algunas estructuras raras. Particularmente, este no separa a X de Y cuando la estructura cumple las dos condiciones siguientes:

Algoritmo 2.3 Algoritmo heurístico para comprobar la necesidad de una arista

```
1: procedimiento EDGENEEDED_H(grafo G, nodos X,Y, conjunto de datos D, umbral ε)
2:     Sea S_X = Ngbr(X) ∩ AdjPath(X,Y) y S_Y = Ngbr(Y) ∩ AdjPath(X,Y)
3:     Eliminar de S_X y S_y todos los nodos hijos de X e Y
4:     para cada conjunto condicionante C ∈ {S_X, S_Y} hacer
5:         Sea s = Ind(X,Y | C)
6:         si s < ε entonces
7:             Hacer CutSet = CutSet ∪ ⟨{X,Y},C⟩
8:             retornar falso
9:         fin si
10:        mientras |C| > 1 hacer
11:            para cada i hacer
12:                Sea C = C \ {el i − esimo nodo de C}
13:                s_i = Ind(X,Y | C_i)
14:            fin para
15:            Sea m = argmin_i {s_1, s_2, ···}
16:            si s_m < ε entonces
17:                retornar falso
18:            sino si s_m > s entonces
19:                Ir al paso 4
20:            sino
21:                Sea s = s_m, C = C_m
22:                Ir al paso 10
23:            fin si
24:        fin mientras
25:    fin para
26:    retornar verdadero
27: fin procedimiento
```

1. Existe al menos un camino de X a Y a través de un hijo de Y y este nodo es del tipo *cabeza-cabeza* ($X \rightarrow Z \leftarrow Y$).
2. Para tales caminos, existe uno o más nodos *cabeza-cabeza* diferente al nodo hijo y todos estos nodos son ancestros (predecesores) de Y.

En tales estructuras, el procedimiento EdgeNeeded_H selecciona, erróneamente, que un padre de Y es un hijo de este nodo y lo elimina del conjunto condicionante. Como resultado, el procedimiento falla en la separación de dos nodos que realmente pueden ser separados. Cheng y col. [26] presentan un ejemplo del fallo de EdgeNeeded_H y la propuesta alternativa EdgeNeeded. También se muestra que EdgeNeeded es correcto para todos los modelos probabilísticos que son monótonamente fiel a un DAG.

La mayor diferencia entre ambos procedimientos es que, además de considerar todos los vecinos de X – llamado S_X – EdgeNeeded también considerará cada vecino de los vecinos de X ($S_X^{'}$). Lo mismo resulta para S_Y y $S_Y^{'}$. Otra diferencia importante es que EdgeNeeded solo considera uno de los conjuntos, ya sea $S_X \cup S_X^{'}$ ó $S_Y \cup S_Y^{'}$; este no tiene que considerar ambos conjuntos.

Ambos algoritmos utilizan una estructura global llamada *CutSet* para almacenar los conjuntos de corte entre los pares de nodos y utilizarla en el proceso de orientación de las aristas.

2.4.3. Orientación del esqueleto

El proceso que utiliza el algoritmo TPDA para orientar el esqueleto de la red Bayesiana es muy similar al que utiliza el LPA. De todos los nodos de una red Bayesiana, solo los que forman patrones *cabeza-cabeza* permiten el paso del flujo de información cuando son instanciados. El mecanismo para identificar las estructuras *cabeza-cabeza* comienza al analizar todas las estructuras de la forma $X - Y - Z$, donde X y Z no están directamente conectados. Existen tres posibles estructuras: (1) $X \rightarrow Y \rightarrow Z$, (2) $X \leftarrow Y \rightarrow Z$, y (3) $X \rightarrow Y \leftarrow Z$, siendo sólo de interés la tercera por permitir el fluyo de información de X a Z cuando Y se instancia, en otras palabras, $Dep(X,Z \mid Y) > Dep(X,Z)$.

Se emplea esta característica de las redes Bayesianas para tratar de orientar la mayor cantidad de aristas utilizando la identificación de los patrones *cabeza-cabeza*. Una limitante de este tipo de procedimiento es que no orienta completamente la red Bayesiana. En un caso extremo, una red que no tiene ninguna estructura *cabeza-cabeza* provoca que no se oriente ninguna arista del grafo.

No obstante a estas limitaciones, estos métodos son populares y muy utilizados en los algoritmos de aprendizaje de redes Bayesianas por su eficiencia y fiabilidad [154, 164]. Existen otros métodos que utilizan procedimientos puros de puntuación [70, 43] para encontrar la dirección correcta de las aristas. Generalmente, son más lentos que los métodos que detectan estructuras *cabeza-cabeza*, producto a que el espacio de búsqueda es más amplio cuando no se tiene un orden a priori de los nodos.

Conflictos en el algoritmo de orientación

La primera dificultad del método de orientación utilizado por TPDA es que no orienta totalmente la red. La segunda dificultad es que en algunos casos se detectan conflictos similares a los que encuentra el LPA [152, 150]. Por último, y no menos importante, el algoritmo de orientación puede crear redes que incumplen la importante condición de ser acíclicas.

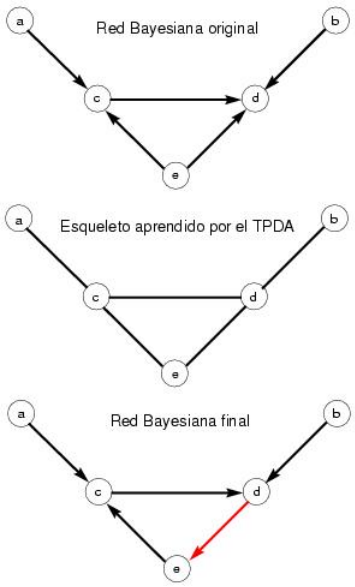

Figura 2.1: Ejemplo del fallo del algoritmo de orientación del TPDA

PROBLEMA 2: El procedimiento *OrientEdges*, utilizado por el TPDA, no orienta completamente la red Bayesiana, sólo crea un grafo esencial de la estructura final.

Para mostrar este problema se parte de la red Bayesiana que se muestra en la parte superior de la figura 2.1 y el esqueleto aprendido por el TPDA (red del centro de la figura 2.1). A continuación se aplican los pasos del algoritmo de orientación propuesto en [26] para orientar el esqueleto.

Inicialmente, se analizan todas las tripletas de la forma $X - Y - Z$, aún no existe ningún arco orientado en la tripleta. En el paso uno del procedimiento *OrientEdges* descrito en [26] se detectan los patrones *cabeza-cabeza* para este tipo de tripleta. El algoritmo comienza analizando la tripleta $c - d - b$, al cumplirse la condición para la creación del patrón *cabeza-cabeza*, se orienta la tripleta de la siguiente forma $c \rightarrow d \leftarrow b$. Posteriormente, se analiza la tripleta $a - c - e$ y el procedimiento la orienta como: $a \rightarrow c \leftarrow e$. Luego de orientar estas dos tripletas este paso no se puede aplicar, pues solo queda por orientar la arista $d - e$. A continuación se ejecuta el paso dos, donde se analizan las tripletas de la forma $X \rightarrow Y - Z$. La idea es no formar nuevos patrones *cabeza-cabeza*, para lo cual si se encuentra una tripleta con la estructura anterior se procede a orientarla como sigue: $X \rightarrow Y \rightarrow Z$. Obsérvese en el ejemplo que la tripleta $b \rightarrow d - e$ cumple con la condición, pues como resultado de aplicar el paso uno, la arista $d - e$ no fue orientada. En este momento el procedimiento toma la decisión de orientar la tripleta de la siguiente forma $b \rightarrow d \rightarrow e$. Esto constituye un grave error al permitir crear el ciclo $c \rightarrow d \rightarrow e \rightarrow c$ (último gráfico de la figura 2.1).

PROBLEMA 3: El procedimiento utilizado por el TPDA infiere, de forma incorrecta, la dirección de los arcos e implica la creación de redes Bayesianas que incumplen la condición de ser acíclicas.

2.4.4. Modificaciones al TPDA Original

A continuación se proponen algunas modificaciones al TPDA para mejorar su eficiencia computacional y lograr insertarlo dentro de la fase de aprendizaje de los algoritmos EDA.

Utilización de la medida de independencia estadística

La primera modificación propuesta al algoritmo se relaciona con el problema uno y es la sustitución de las pruebas de independencia que utilizan criterios de la teoría de la información por la medida estadística que utiliza la prueba χ^2 (sección 1.3.1). La primera razón es la dificultad de obtener un nivel de significación adecuado para el parámetro umbral que controla la independencia (ε). Soto [150] presenta un procedimiento para el cómputo de este parámetro que puede ser ejecutado de forma automática por el algoritmo a costa de aumentar el tiempo de ejecución del mismo. La otra razón es la comparación del algoritmo MMHC con dos variantes del TPDA [55], una que realiza las pruebas de independencia mediante la teoría de la información y otra que utiliza medidas estadísticas. Los resultados muestran que el MMHC supera al TPDA, pero en el caso que utiliza medidas de dependencia estadísticas los resultados son menos significativos. Esto se relaciona con la utilización de un umbral ε no acorde al tamaño de la muestra que se utiliza para el aprendizaje.

Utilización de un método de puntuación en la orientación del esqueleto

Esta modificación se relaciona con los problemas 2 y 3, consiste en sustituir el procedimiento de orientación de las aristas de la red Bayesiana. El algoritmo original utiliza un método basado en restricciones para orientar los arcos, fundamentalmente se dedica a la detección de patrones *cabeza-cabeza.*

Motivados por estos tres problemas y por los excelentes resultados obtenidos por otros algoritmos híbridos como el LPA y el MMHC [114, 115, 162, 150, 20, 21] es que se tomó la decisión de incluir un procedimiento de puntuación para orientar los arcos del esqueleto aprendido por el TPDA. También es importante señalar que la limitación del costo computacional de los algoritmos de puntuación se ve reducida pues su aplicación se restringe al conjunto de posibles aristas a partir del conjunto candidato de padres e hijos.

2.5. Del TPDA al Máx-Mín con Escalador de Colinas

El algoritmo Máx-Mín con Escalador de Colinas (MMHC) es un procedimiento de dos etapas que combina los métodos basados en restricciones y los de puntuación para el aprendizaje de redes Bayesianas. En la primera etapa, el algoritmo descubre la colección de conjuntos candidatos $PC(X)$, los que contienen los padres e hijos de cada variable X. En la segunda etapa, este ejecuta un método de búsqueda local –restringido por los conjuntos $PC(X)$– para orientar la red Bayesiana buscando la solución en el espacio de las redes acíclicas dirigidas.

De acuerdo a las evaluaciones experimentales reportadas, el aprendizaje del MMHC supera, en promedio, a sus predecesores con respecto a la eficiencia computacional, escalabilidad y calidad de las redes Bayesianas aprendidas. A continuación se presenta una breve panorámica del algoritmo; se refiere al lector interesado a los trabajos [20, 21, 163] y a sus referencias para un detallado tratamiento del tópico.

Algoritmo 2.4 Máx-Mín Padres e Hijos

1: **procedimiento** MMPC($T \in X, conjunto\,de\,datos\,D$))
2: **Fase uno (hacia adelante)**
3: $PC(T) = \emptyset$
4: **repetir**
5: $PC(T) = PC(T) \bigcup X$ (donde X maximiza una función heurística basada en pruebas de independencia)
6: **hasta** todas las variables son condicionalmente independientes de T dado algún subconjunto de $PC(T)$
7: **Fase dos (hacia atrás)**
8: Eliminar de $PC(T)$ cualquier variable independiente de T dado algún subconjunto de $PC(T)$
9: **retornar** $PC(T)$
10: **fin procedimiento**

La construcción del conjunto candidato se desarrolla al utilizar un algoritmo local para el descubrimiento causal llamado *Max-Min Parents and Childrens* (MMPC, algoritmo 2.4).

Definición 8. ***(Asociación Mínima)*** *La asociación mínima entre X y T relativa a un subconjunto de rasgos Z, denotada como $MinAssoc(X,T \mid Z)$, se define como:*

$$MinAssoc(X,T \mid Z) = \min_{S \subseteq Z} Assoc(X,T \mid S)$$

La definición anterior expresa la asociación mínima entre X y T sobre todos los subconjuntos de Z. $Assoc(X,T \mid S)$ es la asociación entre dos variables dado un conjunto condicionante. Se implementa utilizando medidas de la teoría de la información o medidas estadísticas. La implementación utiliza la prueba de independencia χ^2 con el estadístico G^2 para decidir la independencia condicional de X_i y X_j dado Z y utiliza el *p-value* negativo de la prueba como la asociación (sección 1.3.1). Esta asociación captura la fuerza de la dependencia con respecto a D y se denota por $Dep(X_i,X_j \mid Z)$ (definición 4).

Dada una variable T y una base de datos de muestras D, el MMPC utiliza un esquema de dos fases para encontrar el conjunto $PC(T)$ [162]. En la fase uno (hacia adelante), las variables se agregan a $PC(T)$ utilizando una función heurística que selecciona la variable que maximiza la asociación (definición 8) con T, condicionado en un subconjunto de la estimación actual de $PC(T)$ y que minimiza dicha asociación (de aquí el nombre del algoritmo). Todas las variables con un arco desde y hacia T y posiblemente otras (falsos positivos) entran en el conjunto $PC(T)$. La fase dos se encarga de eliminar los falsos positivos.

Algoritmo 2.5 Máx-Mín con Escalador de Colinas

```
1:  procedimiento MMHC(conjuntodedatosD)
2:      para cada X_i ∈ X hacer
3:          PC(X_i) = MMPC(X_i, D)
4:      fin para
5:      BN = {}
6:      repetir
7:          Mediante un método escalador de colinas, aplicar los operadores
8:          DeleteArc, InvertArc y AddArc (X → Y se puede adicionar solo si
9:          Y ∈ PC(X))
10:     hasta no se puedan obtener mejoras a la BN actual
11:     retornar BN
12: fin procedimiento
```

Una vez detectado el esqueleto de la red Bayesiana, mediante sucesivas ejecuciones del procedimiento MMPC, se procede a la orientación del mismo. Para inducir la dirección, el MMHC comienza con una red totalmente desconectada y utiliza un método de ascenso a la colina para encontrar la red Bayesiana que optimiza la métrica BIC. El MMHC utiliza tres operadores locales: *eliminar arco* (*DeleteArc*),

invertir arco (*InvertArc*) y *adicionar arco* (*AddArc*). La aplicación del operador *AddArc* se restringe para adicionar solo los arcos $X \rightarrow Y$ donde $Y \in PC(X)$.

2.6. Comparación del MMHC con el TPDA

El MMHC y el TPDA son similares a un nivel fundamental. Aunque el TPDA está compuesto de tres fases o etapas, las dos primeras son similares a la primera fase del MMHC. Esto quiere decir, que ambos algoritmos construyen los conjuntos candidatos de cada no T. En el caso del MMHC es el conjunto *PC*, y en el caso del TPDA es el conjunto de vecinos del nodo T, en el grafo que se tiene construido hasta ese momento. En la tercera fase del TPDA y la segunda del MMHC ambos algoritmos encuentran un conjunto Z, de tal forma que se cumpla la condición $Ind(X, T \mid Z) = 0$. En otras palabras, X se elimina del conjunto de padres e hijos de T en el caso del MMHC y en el TPDA se elimina el arco $T - X$.

La mayor diferencia entre ambos algoritmos es que el MMHC construye el conjunto Z a partir del conjunto vacío y va agregando nuevos nodos. En contraste, el TPDA comienza condicionando en el conjunto completo de padres e hijos y va eliminando nodos hasta que encuentre el conjunto Z.

Cuando el tamaño de los datos no es suficientemente grande para condicionar el conjunto completo de posibles padres e hijos, se espera que la estrategia del TPDA falle en la estimación de la información mutua condicional entre T y cualquier otro nodo. Sin embargo, cuando la muestra es suficientemente grande, el TPDA realiza menos pruebas de independencia para conjuntos condicionantes de varios nodos. Esto se debe, a la estrategia que sigue y a la presencia de suficientes datos para estimar con precisión la información mutua condicional. Estas intuiciones se demuestran empíricamente en varios trabajos [21, 55].

Otra diferencia fundamental es la forma en que se realizan las pruebas de independencia. En el caso del TPDA utiliza la teoría de la información con un umbral ε para la prueba de independencia. Por su parte el MMHC utiliza la prueba χ^2 para determinar si dos variables son independientes (sección 1.3.1).

Brown y col. [21] brindan argumentos sobre la ventaja de utilizar métodos de puntuación en la orientación del esqueleto aprendido por los métodos basados en restricciones. De forma general, el TPDA comienza la fase de orientación con mejores esqueletos que el MMHC para muestras grandes (más de 5000 ejemplos). Sin embargo, la red Bayesiana resultante tiene más errores estructurales que la red de salida del MMHC. Esto indica que los algoritmos basados en restricciones son más eficientes en tiempo de ejecución e identifican un esqueleto de gran calidad, pero son más erráticos en la orientación de la red.

2.7. Del MMHC al CMMHC

En esta sección se introducen las dos modificaciones que se le realiza al algoritmo MMHC para utilizarlo en dominio continuo (llamado CMMHC). Inicialmente se aborda la prueba de Fisher para

detectar la independencia entre dos variables. Posteriormente se introduce la métrica BIC continua para la orientación del esqueleto de la red Gaussiana.

2.7.1. Prueba de Fisher para calcular la independencia condicional

La prueba de independencia entre dos variable continuas se realiza mediante la prueba de Fisher. Esta prueba produce la independencia entre dos variables X_i y X_j, dado un conjunto de variables S, si el coeficiente de correlación parcial ρ es cero. La Z de Fisher [108] se calcula mediante la siguiente ecuación:

$$Z = \frac{1}{2}\sqrt{N - |S| - 3}\left(ln\frac{1+\rho}{1-\rho}\right)$$

donde N es el tamaño de la muestra, $|S|$ es el número de variables en el conjunto condicionante, y ρ es el coeficiente de correlación parcial de X_i y X_j dado S. Para determinar si ρ es cero, se sustituye ρ por cero en la ecuación anterior y se calcula Z. Utilizando la tabla de la distribución normal estándar, podemos determinar la probabilidad que la normal estándar sea mayor que el valor de Z calculado. Si la probabilidad es menor que el nivel de significación α, entonces se rechaza la hipótesis de independencia condicional. En otras palabras, podemos decir que X y Y son condicionalmente dependientes dado S.

2.7.2. Criterio de información Bayesiana continuo

Dada la estructura S y una base de datos D (conjunto seleccionado), la métrica *BIC* continua puede ser expresada como:

$$BIC(S,D) = \sum_{r=1}^{N}\sum_{i=1}^{n}\left[-ln(\sqrt{2\pi v_i}) - \frac{1}{2v_i}(x_{ir} - m_i - \sum_{x_j \in Pa_i} b_{ji}(x_{jr} - m_j))^2\right] - \frac{1}{2}\cdot log(N)\cdot(2n + \sum_{i=1}^{n}|Pa_i|)$$

donde n es el número de variable de la red Gaussiana, N es la cantidad de casos en la base, v_i es la varianza condicional para cada variable X_i dado sus padres Pa_i, m_i es la media de la variable X_i y b_{ji} es el coeficiente de regresión para cada variable X_i en Pa_i.

Al igual que la BIC discreta, una propiedad importante de esta métrica es que se descompone, es decir, se calcula como una suma de las BIC locales de cada variable ($BIC(i,S,D)$).

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

donde

$$BIC(i,S,D) = \sum_{r=1}^{N}\left[-ln(\sqrt{2\pi v_i}) - \frac{1}{2v_i}(x_{ir} - m_i - \sum_{x_j \in Pa_i} b_{ji}(x_{jr} - m_j))^2\right] - \frac{1}{2}\cdot log(N)\cdot(2 + |Pa_i|)$$

2.8. Nuevos EDA basados en pruebas de independencia

En esta sección se presenta un optimizador EDA que utiliza pruebas de independencia y optimización de métricas en el aprendizaje. El mismo se basa en los algoritmos de aprendizajes TPDA, MMHC y CMMHC. Los resultados de estos algoritmos demuestran que son eficientes en la resolución de problemas de aprendizaje automatizado. Esto conduce a disponer de tres EDA, el $\text{CBEDA}_{\text{TPDA}}$, el $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{CMMHC}}$.

La versión simple del CBEDA se muestra en el algoritmo 2.6. Entre otras cosas, este esquema no incluye mutación ni elitismo.

Algoritmo 2.6 CBEDA Simple

Poner $t \leftarrow 1$
Generar $N \gg 0$ puntos de forma aleatoria
mientras no se cumpla el criterio de parada **hacer**
    Evaluar la población en la función $f(x)$
    De acuerdo a un método de selección, construir un conjunto CS de M puntos
    Encontrar la estructura de la red Bayesiana: $BN = \{TPDA(\mathbf{CS}), MMHC(\mathbf{CS}), CMMHC(\mathbf{CS})\}$
    Estimar los parámetros de $p^{CS}(x,t)$ utilizando a BN y **CS**
    Generar N nuevos puntos a partir de $p(x,t+1) \approx p^{CS}(x,t)$
    Poner $t \leftarrow t+1$
fin mientras

2.9. Resultados experimentales

En esta sección se muestran los resultados experimentales que comparan las diferentes propuestas CBEDA con otros algoritmos del estado del arte. Para realizar las pruebas computacionales, a cada algoritmo se le especificaron un conjunto de parámetros comunes. El tamaño de la población seleccionada siempre será del 30 % de la población global, excepto en algún experimento que se especifique otro valor. El método de selección que se utiliza es el *truncamiento*, donde los individuos seleccionados son los mejores de la población global. En ninguno de los experimentos se utilizó el elitismo, se genera la misma cantidad de individuos de la población global. Siempre se realizan 100 corridas de los algoritmos sobre una misma función para promediar los resultados, a menos que se especifiquen otras condiciones.

En todos los experimentos se buscó el *tamaño de población crítica*, esto es, el mínimo de individuos necesarios para alcanzar el óptimo de la función, en al menos, el 95 % de las corridas del algoritmo.

En el caso del $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{TPDA}}$ se utilizó un umbral de 0,05 para la prueba χ^2, lo que se traduce en un 95 % de confidencialidad para la detención de la dependencia entre cualquier par de variables.

2.9.1. Importancia de la orientación en la optimización

La motivación fundamental de este experimento está en los problemas encontrados en la orientación del esqueleto por parte del algoritmo TPDA (sección 2.4.3) y en una idea propuesta en [111, 110, 130].

Los autores implementan una heurística consistente en realizar el aprendizaje a partir de una permutación aleatoria de las variables y posteriormente aprender la estructura Bayesiana del problema utilizando dicho orden. La generación de un orden aleatorio de las variables del problema restringe la cantidad de estructuras que se encuentran por el algoritmo de aprendizaje. La conjetura es que: para una gran cantidad de problemas, la estimación del orden correcto es importante en el proceso de búsqueda de la solución. Para los problemas que se analizan por los autores la heurística funciona de forma satisfactoria [111, 110, 130].

La metodología que se utilizó fue encontrar una B-función con estructura y parámetros determinados, de tal forma, que el FDA no la optimiza satisfactoriamente para todos los órdenes aleatorios generados. La B-función se ha denominado *FirstSingleParent* y tiene una estructura de bosque de poliárboles de seis nodos. La entropía univariada de los padres es baja ($p(x_i = 1) = 0{,}2$) y la de los hijos alta ($p(x_i = 1) = 0{,}45$), además la información mutua de las aristas está entre 0.2 y 0.5. Los experimentos se realizan con 108 variables, es decir, la B-función es un bosque con 18 poliárboles. Dicha función se factoriza de la siguiente manera:

$$p(x) = \prod_{i=0}^{m-1} \left[p(x_{(k+1)\cdot(i+1)}) \prod_{j=1}^{k} p(x_{(k+1)\cdot i+j} \mid p(x_{(k+1)\cdot(i+1)}) \right]$$

donde, m es la cantidad de poliárboles que forman la B-función y k la cantidad de hijos que tiene el último nodo en cada poliárbol.

Al realizar los experimentos se encontró que el tamaño de la población crítica es de 500 individuos. Posteriormente, se generan 100 órdenes aleatorios de las variables y se ejecuta el FDA con dichas factorizaciones.

La tabla 2.1 muestra los resultados en la optimización de la B-función *FirstSingleParent* con el FDA utilizando la factorización correcta y con la factorización completamente invertida, es decir, todos los hijos están orientados hacia el padre.

	Éxitos (%)	Evaluaciones
$FDA_{ORIGINAL}$	$99{,}0 \pm 1{,}59$	$5161{,}62 \pm 43{,}59$
$FDA_{INVERTIDA}$	$15{,}0 \pm 5{,}86$	$8033{,}33 \pm 163{,}48$

Tabla 2.1: Optimización de la B-función *FirstSingleParent* utilizando el FDA con la factorización exacta y con la factorización invertida

La figura 2.2 muestra la razón de éxito para los 100 órdenes aleatorios generados. Los resultados demuestran la conjetura con relación a la importancia de tener una correcta orientación de la red Bayesiana desde el punto de vista de la optimización. Posteriormente en la sección 2.9.4 se retoma el tema del ordenamiento aleatorio, pero en este caso utilizando aprendizaje y no la factorización de la B-función.

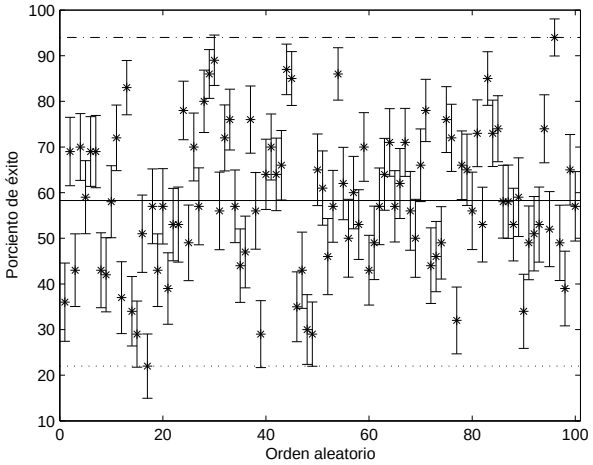

Figura 2.2: Porciento de convergencia del *FDA* para 100 órdenes aleatorios de las variables para la B-función *FirstSingleParent*

2.9.2. Experimento con la B-función *TenLittleNiggers*

Para este experimento se buscó una B-función donde cada nodo hijo tuviese varios padres (nueve), que la entropía univariada de cada uno fuese alta ($p(x_i = 1) = 0{,}49$) y la información mutua de cada arista entre 0.2 y 0.5. Posteriormente se optimiza la función con los algoritmos $\text{CBEDA}_{\text{MMHC}}$, $\text{CBEDA}_{\text{TPDA}}$, EBNA y BOA para comparar el número de evaluaciones. La B-función tiene una estructura de bosque de poliárboles de diez nodos y factoriza como:

$$p(x) = p(x_{10} \mid x_1, x_2 \cdots, x_9) \cdot p(x_{20} \mid x_{11}, x_{12} \cdots, x_{19}) \cdots p(x_{50} \mid x_{41}, x_{42} \cdots, x_{49})$$

En la tabla 2.2 se muestran los resultados de la optimización de la B-función *TenLittleNiggers*. BOA no aparece en la tabla por no lograr optimizar la función con un tamaño de población de 10000 individuos y un máximo de 15 vecinos por nodo. En los casos de los algoritmos $\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{TPDA}}$ el tamaño de población es el crítico, no siendo así para el EBNA donde el experimento se detuvo cuando el tamaño de población fue de 10000 individuos, al igual que con el BOA.

Obsérvese que el MMHC es el que menos evaluaciones realiza, demostrando que tanto el $\text{CBEDA}_{\text{MHHC}}$ como el $\text{CBEDA}_{\text{TPDA}}$ tienen su espacio en la optimización. El resultado más interesante es que los algoritmos basados en pruebas de independencia tienen un mayor porciento de éxito que el EBNA y realizan, considerablemente, menos evaluaciones.

Algoritmo	Tamaño de población	Éxitos	evaluaciones
$\text{CBEDA}_{\text{MMHC}}$	3000	$96{,}0 \pm 4{,}36$	$19625{,}0 \pm 384{,}1$
$\text{CBEDA}_{\text{TPDA}}$	4200	$96{,}0 \pm 4{,}32$	$31937{,}5 \pm 1012{,}4$
EBNA	10000	$84{,}0 \pm 8{,}53$	157381 ± 17275

Tabla 2.2: Resultados para la B-función *TenLittleNiggers*

2.9.3. Experimentos con la subclase RBUF

A continuación se realizan una serie de experimentos con la subclase de B-funciones **RBUF,** enfocados a comparar las propuestas basadas en restricciones con otros algoritmos del estado del arte. Estas funciones no tienen dependencias entre las variables por lo que no tienen arcos en su estructura gráfica.

Experimentos con la subclase BF2B100s0-0045

El objetivo de este experimento es comparar los EDA propuestos con otros del estado del arte sobre la clase de B-funciones que no considera dependencia entre las variables. Primero se detecta el tamaño de población crítica que necesitan los algoritmos UMDA, $\text{CBEDA}_{\text{MMHC}}$, $\text{CBEDA}_{\text{TPDA}}$, EBNA y BOA (con un máximo de uno y tres padres por variable). Posteriormente encontrar el porciento de éxitos para cada algoritmo, la generación de convergencia y la cantidad de evaluaciones. Cada función RBUF está formada por 100 variables, todas con alta entropía univariada ($p(x_i = 1) = 0{,}45$). Se generan 10 funciones RBUF y se realizan 100 corridas por función, lo que implica que los resultados son el promedio de 1000 corridas.

La tabla 2.3 muestra los resultados de la optimización de la RBUF. El UMDA es el mejor algoritmo que optimiza la subclase RBUF, lo que es lógico por ser funciones sin dependencias entre las variables. De los algoritmos que aprenden redes generales, el de mejor resultado es el $\text{CBEDA}_{\text{TPDA}}$ que realiza 6795,88 evaluaciones como promedio, resultado que supera al $\text{CBEDA}_{\text{MMHC}}$, BOA y EBNA.

Un resultado interesante lo constituye el tamaño de población crítica del EBNA que es de 150 individuos, 100 menos que el $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{TPDA}}$. Sin embargo, estos últimos realizan menos evaluaciones, demostrando que los algoritmos basados en restricciones necesitan menos generaciones para encontrar la estructura de la B-función. Por el contrario, el EBNA necesita un mayor número de generaciones, traduciéndose en una mayor cantidad de evaluaciones de la función objetivo.

	Pob. Crítica	Éxitos (%)	Generaciones	Evaluaciones
UMDA	170	$95{,}20 \pm 1{,}06$	$14{,}74 \pm 0{,}06$	$2506{,}61 \pm 11{,}30$
$\text{CBEDA}_{\text{MMHC}}$	500	$96{,}50 \pm 0{,}90$	$13{,}76 \pm 0{,}05$	$6865{,}28 \pm 29{,}64$
$\text{CBEDA}_{\text{TPDA}}$	500	$97{,}00 \pm 0{,}87$	$13{,}59 \pm 0{,}05$	$6795{,}88 \pm 29{,}97$
BOA(k = 1)	700	$96{,}7 \pm 0{,}67$	$13{,}14 \pm 0{,}04$	$9613{,}03 \pm 98{,}96$
BOA(k = 3)	1400	$96{,}05 \pm 0{,}69$	$13{,}46 \pm 0{,}04$	$20431{,}00 \pm 209{,}56$
EBNA	250	$98{,}20 \pm 0{,}71$	$28{,}47 \pm 0{,}23$	$7368{,}38 \pm 57{,}94$

Tabla 2.3: Experimento con 10 funciones de la subclase RBUF: **BF2B100s0-0045**

Escalabilidad con la subclase BF2B*n*s0-0034 de las RBUF

Para este experimento se generaron 30 instancias aleatorias de la B-función **BF2Bns0-0034**, donde n toma los valores $15, 30, 45, 60, y75$. Se buscó el tamaño de población crítica para el cual se

obtuvo el óptimo en las 30 corridas. La figura 2.3 muestra el estudio de escalabilidad para los tres algoritmos ($\text{CBEDA}_{\text{MMHC}}$, BOA y EBNA). Como se observa, los algoritmos BOA $(\text{k} = 1)$, EBNA y $\text{CBEDA}_{\text{MMHC}}$ tienen una escalabilidad similar aunque el $\text{CBEDA}_{\text{MMHC}}$ realiza en casi todos los casos (excepto $n = 45$) menos evaluaciones que el resto de los algoritmos.

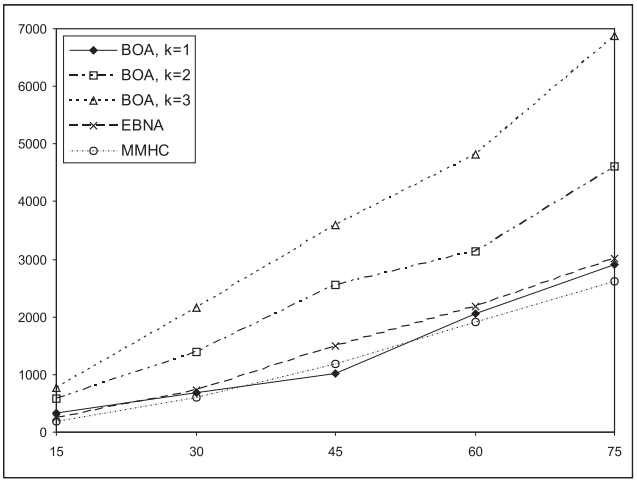

Figura 2.3: Escalabilidad promedio de los algoritmos $\text{CBEDA}_{\text{MMHC}}$, BOA y EBNA sobre 30 instancias de la subclase **RBUF**

2.9.4. Calidad del modelo aprendido

En la tabla 2.4 se muestran los resultados de la optimización de la B-función **BF2B30s3-3445** con tres variantes del algoritmo $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{TPDA}}$. El primer algoritmo es el clásico descrito en la sección 2.8, donde se construye inicialmente un esqueleto del modelo y posteriormente se orienta siguiendo un método de optimización de métrica. La segunda variante sustituye el paso de orientación por métrica, por un proceso de orientación aleatoria ($\text{CBEDA}_{\text{MMHC}-\text{RO}}$). Por último, se utiliza un algoritmo MMHC que construye un poliárbol ($\text{CBEDA}_{\text{MMHC}-\text{PT}}$). La B-función consta de 30 variables donde cada una tiene como máximo tres padres, alto valor de dependencia entre las variables (entre 0.3 y 0.4) y una alta entropía univariada ($p(x_i = 1) = 0{,}45$). El tamaño de población que se utiliza es de 2500 individuos. Para cada algoritmo se muestra en la tabla el porciento de éxitos alcanzados, y la cantidad de evaluaciones realizadas. A continuación las próximas tres columnas están relacionadas con la calidad del modelo aprendido. Primero se examina la cantidad de arcos que coinciden en la estructura aprendida en la última generación del algoritmo con la estructura de la B-función. La próxima columna representa la cantidad de arcos que están invertidos. La última columna muestra la cantidad de arcos que añade de más el algoritmo. Los resultados son el promedio de 100 corridas.

	Éxitos (%)	Evaluaciones	Correctos	Invertidos	Excedente
$CBEDA_{MMHC}$	$99{,}00 \pm 1{,}59$	$9368{,}69 \pm 274{,}22$	$13{,}48 \pm 0{,}30$	$14{,}51 \pm 0{,}29$	$10{,}32 \pm 0{,}64$
$CBEDA_{MMHC-RO}$[6]	$47{,}00 \pm 8{,}21$	$11861{,}70 \pm 754{,}90$	$14{,}04 \pm 0{,}72$	$13{,}95 \pm 0{,}75$	$34{,}04 \pm 1{,}49$
$CBEDA_{MMHC-PT}$[7]	$100{,}00 \pm 0$	$9575{,}00 \pm 316{,}50$	$13{,}23 \pm 0{,}31$	$14{,}68 \pm 0{,}30$	$1{,}02 \pm 0{,}19$
$CBEDA_{TPDA}$	$98{,}00 \pm 2{,}27$	$9183{,}67 \pm 298{,}81$	$13{,}83 \pm 0{,}32$	$14{,}16 \pm 0{,}27$	$10{,}13 \pm 0{,}56$

Tabla 2.4: Calidad del modelo aprendido por los CBEDA

Los mejores resultados los alcanzan el $\text{CBEDA}_{\text{TPDA}}$, el $\text{CBEDA}_{\text{MMHC}}$ clásico y el que aprende una estructura de poliárbol. El $\text{CBEDA}_{\text{MMHC-PT}}$ es el que obtiene el modelo más cercano a la estructura de la B-función, apenas agrega solo un arco de más, como promedio. En este análisis se conjetura que los arcos que están bien orientados son los que pertenecen a las estructuras *cabeza-cabeza* por lo que el modelo probabilístico puede ser el mismo aunque hayan varios arcos invertidos. Por último, se muestra que disponer de un proceso correcto de orientación es fundamental para la optimización. En el caso del $\text{CBEDA}_{\text{MMHC-RO}}$ es el que peores resultados obtiene con solo el 47 % de convergencia. Esto se debe a la cantidad de arcos que fueron añadidos por el algoritmo de forma innecesaria.

2.9.5. Importancia de la estructura en la optimización

En esta sección se estudia la importancia de tener una factorización lo más cercana posible a la real de la función objetivo. El estudio se divide en dos partes fundamentales, primero se estudia el algoritmo FDA con la función F_{trap}. Posteriormente, el caso de los algoritmos que utilizan aprendizaje, específicamente el $\text{CBEDA}_{\text{MMHC}}$.

FDA y la estructura de la función F_{trap}

La motivación de este experimento se encuentra en los planteamientos encontrados en [78]. Los autores utilizan para comprobar su hipótesis la función F_{trap} (ecuación A.7). En el referido artículo se expone: *"para que el algoritmo BOA realice un aprendizaje correcto de la estructura de enlace, al menos una de las variables que forman la sub-función debe ser dependiente del resto de las variables de la misma sub-función"*. Según este planteamiento, si se consideran las factorizaciones de la forma: $X_{5i-4} \leftarrow X_{5i-3}, X_{5i-2}, X_{5i-1}, X_{5i}$, bastaría para que el algoritmo BOA encuentre el óptimo para la función F_{trap}. El primer experimento está encaminado a refutar esta propuesta mediante la ejecución del algoritmo FDA con la factorización antes mencionada e ir adicionando aristas para estudiar el comportamiento en la optimización.

Para efectuar el experimento se realizan 100 corridas del FDA con cada factorización. La condición de parada es encontrar el óptimo (éxito) o alcanzar la generación 30. Se parte de una población inicial de 100 individuos y se incrementa hasta alcanzar los 1000 individuos, en tal caso se considera que el FDA no es capaz de optimizar la función.

Para la factorización inicial el FDA no es capaz de encontrar el óptimo en ninguna de las corridas, pero no solo esto, sino que cuando se agregan las aristas: $X_{5i-3} \leftarrow X_{5i-2}$, $X_{5i-3} \leftarrow X_{5i-1}$, $X_{5i-3} \leftarrow X_{5i}$ el algoritmo también falla en el 100 % de las corridas. La tabla 2.5 muestra los resultados a partir

de la primera arista que al añadirse no falla la optimización en todas las corridas. Otro resultado, corroborado con anterioridad por Ochoa y Soto [118], es la importancia de tener la estructura del EDA lo más cercana a la real de la función. En el experimento se aprecia cómo la estructura más certera es la que considera la factorización completa, lo que es de esperar al diseñar el experimento.

Arista	N	Éxitos (%)	Evaluaciones
$X_{5i-2} \leftarrow X_{5i-1}$	1000	$68{,}0 \pm 7{,}81$	$8617{,}65 \pm 252{,}63$
$X_{5i-2} \leftarrow X_{5i}$	300	$97{,}0 \pm 2{,}74$	$1948{,}45 \pm 58{,}06$
$X_{5i-1} \leftarrow X_{5i}$	**250**	$96{,}0 \pm 3{,}37$	$\mathbf{1343{,}75 \pm 34{,}76}$

Tabla 2.5: Comportamiento del algoritmo FDA a medida que se añaden nuevas aristas a la estructura de la función F_{trap}

A partir de los resultados, se plantea la conjetura de que los algoritmos que utilizan aprendizaje encuentran una estructura lo más cercana posible a la factorización real de la función objetivo para no fallar en el proceso de optimización. Este hecho es objeto de estudio en las próximas dos secciones.

$\text{CBEDA}_{\text{MMHC}}$ y la estructura de la función F_{trap}

En la sección anterior se demostró la importancia que tiene la correcta identificación de la estructura de enlace para el FDA. En este apartado se analiza el problema cuando se utilizan algoritmos que realizan aprendizaje de la estructura como el $\text{CBEDA}_{\text{MMHC}}$.

La configuración del experimento consiste en la optimización de la función Trap (fórmula A.7 en anexo A) con 30 variables. Primero se obtuvo el tamaño de población crítica para un 95 % de convergencia. En cada corrida, cuando se encuentra el óptimo se guarda la estructura descubierta por el algoritmo (matriz de adyacencia). Posteriormente, se sumaron todas las matrices de adyacencia sin tener en cuenta la dirección de los arcos. Los resultados se muestran en la figura 2.4, mientras más acentuado es el nivel de gris, más veces aparece la arista en la matriz de adyacencia. Se observa que las aristas se concentran alrededor de los clústeres de variables. Esta es una función ideal para el experimento pues al ser separable se aprecia bien su estructura.

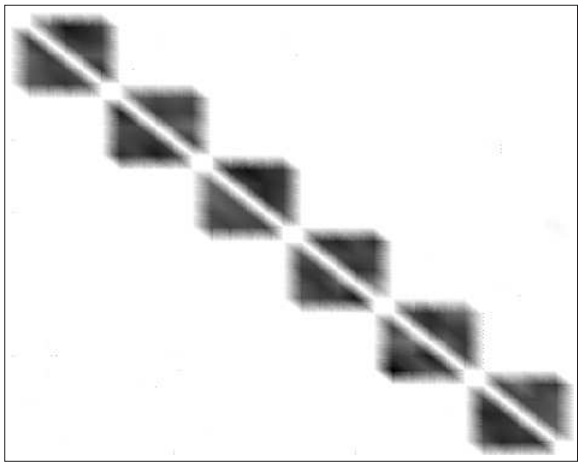

Figura 2.4: Matriz de adyacencia encontrada por el $\text{CBEDA}_{\text{MMHC}}$ para la función F_{trap}

Este resultado muestra la conjetura del apartado anterior sobre la importancia que tiene en la optimización que los algoritmos de aprendizaje encuentren una estructura lo más cercana a la original de la función a optimizar.

En la próxima sección se muestra un estudio similar con la B-función BF2B30s3-1212.

$\text{CBEDA}_{\text{MMHC}}$ y la estructura de la función BF2B30s3-1212

La metodología utilizada en este experimento es similar a la del apartado anterior. La figura 2.5 refleja las matrices de adyacencias encontradas por el $\text{CBEDA}_{\text{MMHC}}$. La figura superior izquierda muestra la estructura original de la B-función, la superior derecha la estructura resultante de la suma de todas las estructuras en las corridas exitosas. Como se aprecia en la estructura, existen arcos no pertenecientes a la B-función que son incluidos, pero por el claro del gris se asume que son muy esporádicos. La figura inferior izquierda representa la estructura al hacer un corte con las aristas que aparecen más de 90 veces en la suma final de las estructuras. Obsérvese que la matriz de adyacencia es similar a la estructura de la B-función (superior izquierda). Por último la figura inferior derecha representa lo mismo pero realizando un corte sobre las 95 aristas, en este caso se excluyen algunas que aparecen entre 90 y 95 porciento de las veces.

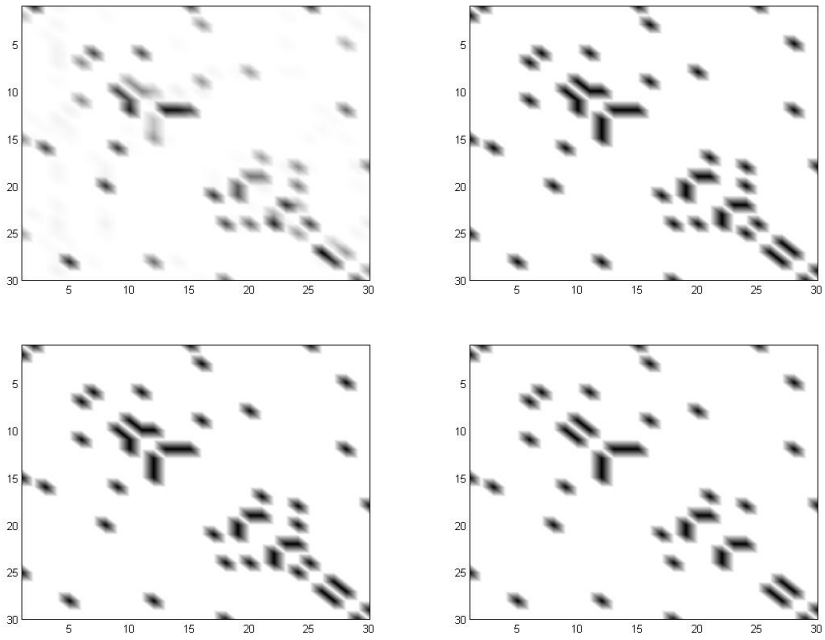

Figura 2.5: Comparación de las matrices de adyacencia de la B-función con las obtenidas por el $CBEDA_{MMHC}$

Como conclusión se asegura que el correcto aprendizaje de la estructura de enlace juega un papel importante en la optimización, tanto para funciones clásicas, de las cuales no se tiene clara su estructura, como para las B-funciones.

2.9.6. El $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{TPDA}}$ optimizan funciones enteras

Una característica importante de los algoritmos propuestos en esta investigación es que permiten optimizar funciones con cardinalidad entera, es decir, las variables toman valores enteros no binarios. Para realizar estos experimentos se utilizó la posibilidad que brinda el generador de B-funciones de especificar la estructura y los parámetros de la B-función.

En un primer experimento se generan ocho B-funciones con la misma estructura y con 20 variables. La cardinalidad varía de dos a nueve y se optimizan con los algoritmos EBNA y $\text{CBEDA}_{\text{MMHC}}$. El EBNA permite optimizar funciones enteras pero con la restricción de que todas las variables deben tener la misma cardinalidad. La implementación actual de los CBEDA permite resolver problemas con cardinalidades mixtas, y esto se muestra en el próximo experimento. En la figura 2.6 se aprecia que hasta la cardinalidad seis, ambos algoritmos tienen un comportamiento similar en cuanto al número de evaluaciones realizadas, con ligera ventaja para el EBNA. A partir de siete, los resultados del EBNA se tornan exponenciales, no siendo así para el $\text{CBEDA}_{\text{MMHC}}$. Una vez más queda demostrada la superioridad del MMHC sobre el EBNA para la función entera estudiada.

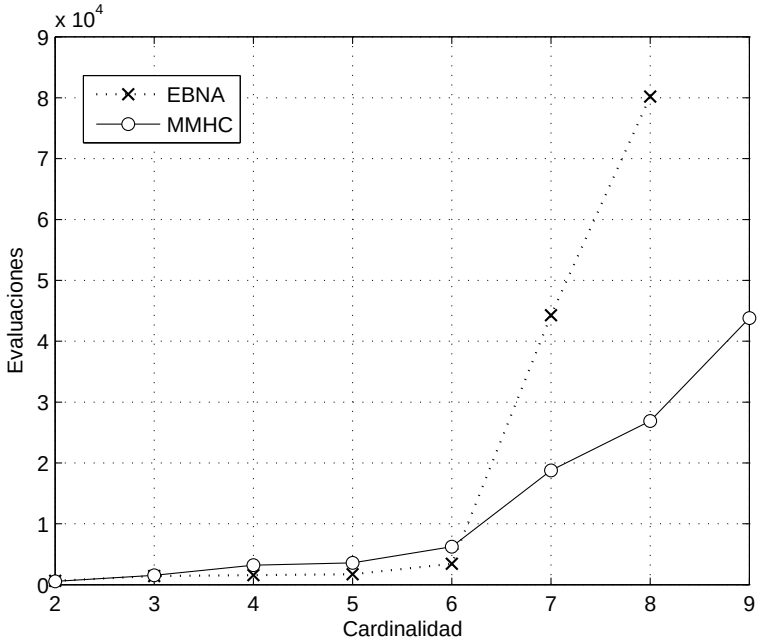

Figura 2.6: Optimización de una B-función entera por los algoritmos EBNA y $\text{CBEDA}_{\text{MMHC}}$

En un segundo momento se utilizan B-funciones generadas a partir de dos redes Bayesianas con estructura conocida (Carpo[8] y S4[9]). Las cardinalidades de las variables en ambas redes, están en el intervalo de dos a cuatro.

En la tabla 2.6 se muestran los resultados de la optimización de ambas funciones Bayesianas. Ambos algoritmos optimizan dichas funciones, con un tamaño de población crítica medio y realizan pocas

[8]http://www.cs.huji.ac.il/~galel/Repository/

[9]http://banquiseasi.insa-rouen.fr/projects/bnt-slp/

evaluaciones. El comportamiento de las dos propuestas es similar para la B-función S4. En la red Carpo, el $\text{CBEDA}_{\text{TPDA}}$ realiza menos evaluaciones que el $\text{CBEDA}_{\text{MMHC}}$, característica presente en experimentos anteriores.

			$\text{CBEDA}_{\text{MMHC}}$		$\text{CBEDA}_{\text{TPDA}}$	
	Vars.	Pob. Crítica	Éxitos (%)	Evaluaciones	Éxitos (%)	Evaluaciones.
Carpo	60	800	$95{,}00 \pm 3{,}43$	$9338{,}95 \pm 114{,}98$	$98{,}00 \pm 2{,}26$	$9159{,}18 \pm 99{,}59$
S4	15	500	$98{,}00 \pm 2{,}42$	$2408{,}16 \pm 71{,}54$	$95{,}00 \pm 3{,}58$	$2436{,}84 \pm 104{,}59$

Tabla 2.6: Experimento con B-funciones de cardinalidad entera basadas en redes Bayesianas

2.9.7. El problema de la predicción de estructuras de proteínas

En esta sección se muestran los resultados de la aplicación del $\text{CBEDA}_{\text{TPDA}}$ en la solución del problema de la predicción de estructuras de proteínas (sección A.4) y su comparación con otros algoritmos del estado del arte de la computación evolutiva.

La tabla 2.7 muestras las instancias del modelo HP utilizadas en los experimentos, $H(x^*)$ representa la mejor solución conocida para la secuencia. PSP se ha estudiado en varios trabajos [31, 33, 143], utilizando las secuencias descritas.

Instancia	tamaño	$H(x^*)$	secuencia
s_1	20	-9	$HPHPPHHPHHPHPHHPPHPH$
s_2	24	-9	$HHPPHPPHPPHPPHPPHPPHPPHH$
s_3	25	-8	$PPHPPHHP^4HHP^4HHP^4HH$
s_4	36	-14	$P^3HHPPHHP^5H^7PPHHP^4HHPPHPP$
s_5	48	-23	$PPHPPHHPPHHP^5H^{10}P^6$ $HHPPHHPPHPPH^5$
s_6	50	-21	$HHPHPHPHPH^4PHP^3HP^3HP^4$ $HP^3HP^3HPH^4\{PH\}^4H$
s_7	60	-36	$PPH^3PH^8P^3H^{10}PHP^3$ $H^{12}P^4H^6PHHPHP$
s_8	64	-42	$H^{12}PHPH\{PPHH\}^2PPH\{PPHH\}^2$ $PPH\{PPHH\}^2PPHPHPH^{12}$

Tabla 2.7: Instancias del modelo HP utilizadas en los experimentos

El primer experimento consiste en encontrar el óptimo para las secuencias de la tabla 2.7 utilizando tres variantes del $\text{CBEDA}_{\text{TPDA}}$. La primera variante es el algoritmo $\text{CBEDA}_{\text{TPDA}}$ clásico, la segunda realiza una orientación aleatoria del esqueleto ($\text{CBEDA}_{\text{TPDA-RO}}$) y la tercera restringe el número de nodos adyacentes a dos ($\text{CBEDA}_{\text{TPDA-k2}}$).

La metodología empleada es similar a la utilizada en el estudio de otros EDA [143]. Todos los algoritmos tienen un tamaño de población de 5000 individuos, ejecutándose un máximo de 5000 generaciones. El truncamiento es de 0.1 con la aplicación del mejor elitismo (toda la población seleccionada pasa directamente a la próxima generación). La tabla 2.8 muestra los resultados obtenidos

en los experimentos, donde *éxitos* es el porciento de veces que se encontró la mejor solución en 50 corridas independientes y *gen* representa la generación media donde se encontró la mejor solución. Los mejores resultados de cada parámetro medido son destacados en negritas.

F		$\text{CBEDA}_{\text{TPDA}}$			$\text{CBEDA}_{\text{TPDA-RO}}$			$\text{CBEDA}_{\text{TPDA-k2}}$		
	$H(x^*)$	$H(x)$	*éxitos*	*gen*	$H(x)$	*éxitos*	*gen*	$H(x)$	*éxitos*	*gen*
s_1	-9	**-9**	**100**	**3.32**	**-9**	**100**	**3.32**	**-9**	**100**	3.34
s_2	-9	**-9**	**100**	**3.66**	**-9**	**100**	3.98	**-9**	**100**	3.74
s_3	-8	**-8**	**100**	**4.52**	**-8**	**100**	4.54	**-8**	96	5.81
s_4	-14	**-14**	4	12.5	**-14**	4	12.5	**-14**	**10**	**12.4**
s_5	-23	**-23**	**6**	**21.0**	-22	8	53	**-23**	4	27.5
s_6	-21	**-21**	**88**	13.86	**-21**	62	16.22	**-21**	76	**13.76**
s_7	-36	-35	16	**54.37**	-35	**20**	79.7	-35	4	144.5
s_8	-42	**-42**	10	**28.0**	**-42**	**24**	50.58	**-42**	4	94.0

Tabla 2.8: Resultados para los algoritmos $\text{CBEDA}_{\text{TPDA}}$, $\text{CBEDA}_{\text{TPDA-RO}}$, $\text{CBEDA}_{\text{TPDA-k2}}$ en la solución de PSP

Los resultados muestran como todas las variantes del $\text{CBEDA}_{\text{TPDA}}$ cuentran la mejor solución para las secuencias $s_1 - s_6$ y s_8 excepto el $\text{CBEDA}_{\text{TPDA-RO}}$ en la secuencia s_5 que obtiene una solución sub-óptima. En el caso de la secuencia s_7 todos los algoritmos obtienen una solución sub-óptima, en [143] se muestra empíricamente como esta instancia es decepcionante lo que aumenta su complejidad. Al igual que en [143] los resultados para los CBEDA son promisorios debido a que no se utilizan técnicas de optimización local ni los parámetros se han refinado para cada instancia.

El siguiente experimento tiene como motivación la siguiente pregunta: ¿Es necesario acudir a modelos de aprendizaje más complejos como las redes de Markov para la solución de PSP tridimensional con EDA? Para dar respuesta a la pregunta seguimos un experimento similar al utilizado por Santana y otros [143], configurando el $\text{CBEDA}_{\text{TPDA}}$ con un tamaño de población de 5000 individuos y un máximo de 1000 generaciones. El truncamiento es de 0.1 con la estrategia del mejor elitismo al igual que en los experimentos para retículos en dos dimensiones. La comparación se realiza entre el GA híbrido [31], el $\text{MK} - \text{EDA}_2$[10] y el $\text{CBEDA}_{\text{TPDA}}$.

	GA híbrido		$\text{MK} - \text{EDA}_2$		$\text{CBEDA}_{\text{TPDA}}$	
	$H(x)$	$media \pm \sigma$	$H(x)$	$media \pm \sigma$	$H(x)$	$media \pm \sigma$
s_1	-11	$-10{,}52 \pm 0{,}54$	-11	$-10{,}82 \pm 0{,}38$	**-11**	$\mathbf{-11{,}0 \pm 0{,}0}$
s_2	-13	$-11{,}28 \pm 0{,}90$	-13	$-12{,}02 \pm 0{,}94$	**-13**	$\mathbf{-13{,}0 \pm 0{,}0}$
s_3	-9	$-8{,}54 \pm 0{,}64$	-9	$-8{,}96 \pm 0{,}19$	**-9**	$\mathbf{-9{,}0 \pm 0{,}0}$
s_4	-18	$-15{,}76 \pm 1{,}05$	-18	$-16{,}40 \pm 0{,}80$	**-18**	$\mathbf{-17{,}96 \pm 0{,}06}$
s_5	-28	$-24{,}60 \pm 1{,}57$	**-29**	$\mathbf{-27{,}24 \pm 0{,}92}$	-29	$-25{,}36 \pm 0{,}37$
s_6	-26	$-23{,}02 \pm 1{,}48$	-29	$-25{,}70 \pm 1{,}26$	**-31**	$\mathbf{28{,}72 \pm 0{,}19}$
s_7	-49	$-41{,}18 \pm 2{,}75$	**-49**	$\mathbf{-46{,}30 \pm 2{,}04}$	-49	$41{,}86 \pm 0{,}36$
s_8	-46	$-40{,}40 \pm 2{,}50$	**-52**	$\mathbf{-46{,}78 \pm 2{,}28}$	-50	$41{,}2 \pm 0{,}64$

Tabla 2.9: Resultados del $\text{CBEDA}_{\text{TPDA}}$, el GA híbrido y el $\text{MK} - \text{EDA}_2$ en retículos tridimensionales

[10]El subíndice dos significa que cada nodo puede tener a lo sumo dos nodos adyacentes

Los resultados se muestran en la tabla 2.9. Se puede apreciar que el $\text{CBEDA}_{\text{TPDA}}$ supera al GA híbrido y al $\text{MK}-\text{EDA}_2$ en el promedio del óptimo obtenido para las instancias de la s_1 a s_4 y para la instancia s_6. En el caso de las instancias s_5 y s_7 ambos EDA obtienen el mismo óptimo aunque el $\text{MK}-\text{EDA}_2$ supera en el promedio del óptimo al $\text{CBEDA}_{\text{TPDA}}$. Para la instancia s_8 el $\text{MK}-\text{EDA}_2$ mejora al GA híbrido y al $\text{CBEDA}_{\text{TPDA}}$, tanto en el óptimo encontrado como en el promedio. Una observación importante es que los EDA superaron siempre los resultados obtenidos por el GA híbrido, aun cuando este último utiliza técnicas de optimización local para la resolución del problema PSP.

Cómo respuesta a la pregunta planteada podemos decir que no parece necesario acudir a modelos complejos de EDA para la solución de PSP en las secuencias estudiadas. Los modelos Bayesianos optienen resultado similares a los modelos Markovianos con la ventaja de no estar restringida la estructura (en el $\text{MK}-\text{EDA}_2$ cada nodo puede tener a lo sumo dos nodos adyacentes). En una futura investigación se puede comprobar si la inclusión de un optimizador local mejora las soluciones obtenidas con el $\text{CBEDA}_{\text{TPDA}}$.

2.9.8. Resultados para dominio continuo

En esta sección se evalúa del rendimiento del algoritmo $\text{CBEDA}_{\text{CMMHC}}$, comparándolo con otros algoritmos del campo de los EDA continuos. También se analizan algunas cuestiones interesantes del funcionamiento del algoritmo como son la asimetría de los intervalos de definición de la función objetivo. Específicamente se estudian tres funciones continuas: la F_{Sphere} (ecuación A.10), la F_{Ackley} (ecuación A.12) y la función $F_{Griewangk}$ (ecuación A.11). Los algoritmos estudiados son: GUMDA, EUMDA, EMNA, GGCEDA y EGCEDA [10]. Los resultados de los algoritmos se toman de dicha publicación.

Para la ejecución de los experimentos el método de selección que se utiliza es el truncamiento con un umbral de 0.3. El algoritmo finaliza si el óptimo se encuentra con un error de 10^{-7}, o si se alcanzan 400000 evaluaciones de la función objetivo. Los problemas están formados por 10 y 50 variables.

Rendimiento del $\text{CBEDA}_{\text{CMMHC}}$

Las tablas 2.10, 2.11 y 2.12 muestran la comparación del algoritmo propuesto, con otros del estado del arte para las tres funciones estudiadas. La columna *N* de las tablas muestra el tamaño de población crítica, seguido de la generación de convergencia (columna generación) que se interpreta de forma similar al número de evaluaciones que realiza el algoritmo (columna evaluaciones). Posteriormente se muestra el porciento de éxitos del algoritmo (siempre tiene que ser superior o igual al 95 %) y el error medio con que se detuvo el algoritmo.

En el caso de la función F_{Sphere} con dimensión 10, el $\text{CBEDA}_{\text{CMMHC}}$ obtiene el menor número de evaluaciones, no siendo así para 50 variables. Aunque el resultado se acerca al alcanzado por el GUMDA, se destaca que el error obtenido es menor que el resto de los algoritmos. Una vez más se pone de manifiesto la no relación existente entre el tamaño de población crítica y el número de evaluaciones realizada por el algoritmo. Esto se aprecia en el caso del EUMDA cuya población crítica

es de 55, sin embargo, el $\text{CBEDA}_{\text{CMMHC}}$ y el GUMDA realizan menor cantidad de evaluaciones. Para las funciones F_{Ackley} y $F_{Grienwangk}$ los resultados se interpretan de forma similar.

Algoritmo	N	Generación	Éxitos	Error	Evaluaciones
$\text{CBEDA}_{\text{CMMHC}}$	80	47	96 %	$6{,}87 \times 10^{-8}$	**3756**
GUMDA	80	47	95 %	$2{,}54 \times 10^{-7}$	3805
EUMDA	55	78	97 %	$4{,}16 \times 10^{-7}$	4329
EMNA	265	43	96 %	$3{,}58 \times 10^{-7}$	11443
GGCEDA	265	43	97 %	$3{,}53 \times 10^{-7}$	11662
EGCEDA	175	62	99 %	$3{,}80 \times 10^{-7}$	11017
$\text{CBEDA}_{\text{CMMHC}}$	180	135	96 %	$8{,}54 \times 10^{-8}$	24473
GUMDA	160	126	95 %	$4{,}41 \times 10^{-7}$	**20250**
EUMDA	55	532	100 %	$4{,}51 \times 10^{-7}$	29264
EMNA	2090	109	96 %	$4{,}16 \times 10^{-7}$	229595
GGCEDA	2080	110	95 %	$4{,}64 \times 10^{-7}$	228975
EGCEDA	1300	164	97 %	$4{,}83 \times 10^{-7}$	213030

Tabla 2.10: Rendimiento de los algoritmos en la función F_{Sphere}

Algoritmo	N	Generación	Éxitos	Error	Evaluaciones
$\text{CBEDA}_{\text{CMMHC}}$	100	63	98 %	$8{,}18 \times 10^{-8}$	6372
GUMDA	85	61	95 %	$2{,}17 \times 10^{-7}$	**5246**
EUMDA	55	103	97 %	$4{,}97 \times 10^{-7}$	5679
EMNA	335	57	95 %	$4{,}77 \times 10^{-7}$	19229
GGCEDA	330	57	96 %	$3{,}71 \times 10^{-7}$	19064
EGCEDA	180	82	97 %	$4{,}85 \times 10^{-7}$	14841
$\text{CBEDA}_{\text{CMMHC}}$	180	164	98 %	$9{,}19 \times 10^{-8}$	29672
GUMDA	170	156	96 %	$4{,}96 \times 10^{-7}$	**26682**
EUMDA	55	667	98 %	$4{,}91 \times 10^{-7}$	36696
EMNA	2500	161	0 %	3,97	>400000
GGCEDA	2500	161	0 %	3,88	>400000
EGCEDA	1700	195	98 %	$4{,}95 \times 10^{-7}$	332451

Tabla 2.11: Rendimiento de los algoritmos continuos en la función F_{Ackley}

Asimetría del intervalo y cercanía al óptimo

En esta sección se estudia dos cuestiones importantes para entender el funcionamiento del $\text{CBEDA}_{\text{CMMHC}}$. Primero se analiza cómo evoluciona la población seleccionada con respecto a la cercanía de sus puntos al óptimo para la función F_{Sphere}. Posteriormente, para la función F_{Ackley} y $F_{Griewangk}$ se analiza la influencia de la variación de los extremos del intervalo de definición de la función en el proceso de optimización.

Algoritmo	N	Generación	Éxitos	Error	Evaluaciones
$\text{CBEDA}_{\text{CMMHC}}$	100	48	98 %	$7{,}01 \times 10^{-8}$	**4853**
GUMDA	105	47	95 %	$3{,}90 \times 10^{-7}$	5030
EUMDA	900	71	96 %	$2{,}98 \times 10^{-7}$	63350
EMNA	310	43	97 %	$2{,}74 \times 10^{-7}$	13413
GGCEDA	305	43	97 %	$1{,}82 \times 10^{-7}$	13174
EGCEDA	400	66	96 %	$3{,}91 \times 10^{-7}$	26425
$\text{CBEDA}_{\text{CMMHC}}$	180	121	97 %	$\times 10^{-7}$	21934
GUMDA	165	111	97 %	$4{,}39 \times 10^{-7}$	**18406**
EUMDA	55	475	100 %	$4{,}62 \times 10^{-7}$	26125
EMNA	2075	97	98 %	$3{,}42 \times 10^{-7}$	202503
GGCEDA	2090	97	97 %	$3{,}69 \times 10^{-7}$	203506
EGCEDA	1285	145	96 %	$4{,}27 \times 10^{-7}$	186620

Tabla 2.12: Rendimiento de los algoritmos continuos en la función $F_{Griewangk}$

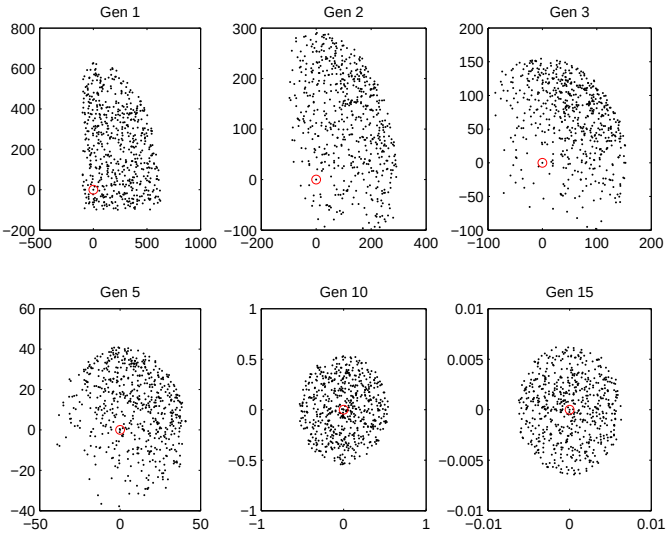

Figura 2.7: Seguimiento de los puntos de la población seleccionada para función F_{Sphere} con dos variables

La figura 2.7 muestra que a medida que aumenta el número de generaciones los puntos del conjunto seleccionado se desplazan hacia el óptimo de la función F_{Sphere}. A partir de la generación cinco (segunda fila, primera figura) los puntos se concentran alrededor del cero, lo que implica la disminución de la varianza (eje de las abscisas). Esto se corrobora en la figura 2.8 que muestra la distribución empírica del conjunto seleccionado (línea continua en negritas) y la distribución normal de dicho conjunto. Ambas distribuciones tienen curvas muy similares con respecto a la media y la varianza.

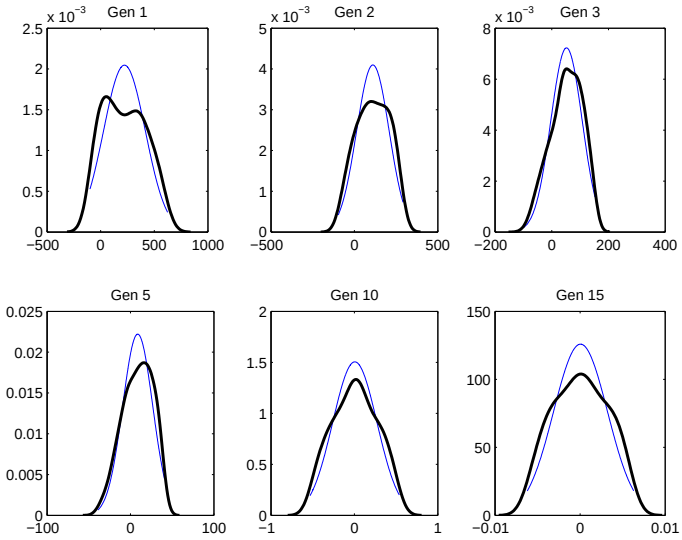

Figura 2.8: Distribución empírica y normal de la primera variable de la función F_{Sphere}

En el próximo experimento se analiza el efecto de correr uno de los extremos del intervalo de definición de las funciones F_{Ackley} y $F_{Griewangk}$, esto se llama *asimetría del intervalo*. La tabla 2.13 muestra que a medida que se acentúa la asimetría aumenta el número de evaluaciones realizadas por los algoritmos. Producto de la cercanía del óptimo (0,0) a uno de los extremos (izquierdo) del intervalo, el $\text{CBEDA}_{\text{CMMHC}}$ necesita mayor cantidad de puntos en la población para hacer una correcta estimación de la distribución de búsqueda, influyendo directamente en el número de evaluaciones realizadas por el algoritmo. De hecho, es el algoritmo de peor comportamiento ante estas situaciones cambiantes en la función.

Intervalos	EUMDA	GUMDA	$\text{CBEDA}_{\text{CMMHC}}$
$[-20,25]$	5709	5367	6377
$[-15,30]$	5772	5328	6332
$[-8,37]$	5831	6360	10026
$[-1,44]$	6884	10881	39317

Tabla 2.13: Comparación de los algoritmos EUMDA, GUMDA, y $\text{CBEDA}_{\text{CMMHC}}$ para diferentes intervalos en la función F_{Ackley} con 10 variables

Para la función $F_{Griewangk}$ se realiza un estudio similar pero desde otra perspectiva, en este caso se sigue el mejor punto encontrado en cada generación de una corrida del $\text{CBEDA}_{\text{CMMHC}}$. Se parte de la población crítica (100 individuos) para solucionar el problema en su intervalo original $[-600,600]$ y se realizan ejecuciones para los intervalos $[-300,600]$, $[-100,600]$, $[-50,600]$, $[-10,600]$ y $[-1,600]$. En un segundo momento, se ajusta el tamaño de población crítica, en cada intervalo, para alcanzar un 95 % de éxito en 100 corridas y se grafican los resultados anteriores. Para hacer más legibles los datos mostrados, se acotan los intervalos del mejor individuo encontrado (eje de la ordenadas) con valores máximos de cinco, dos, uno y 0.01.

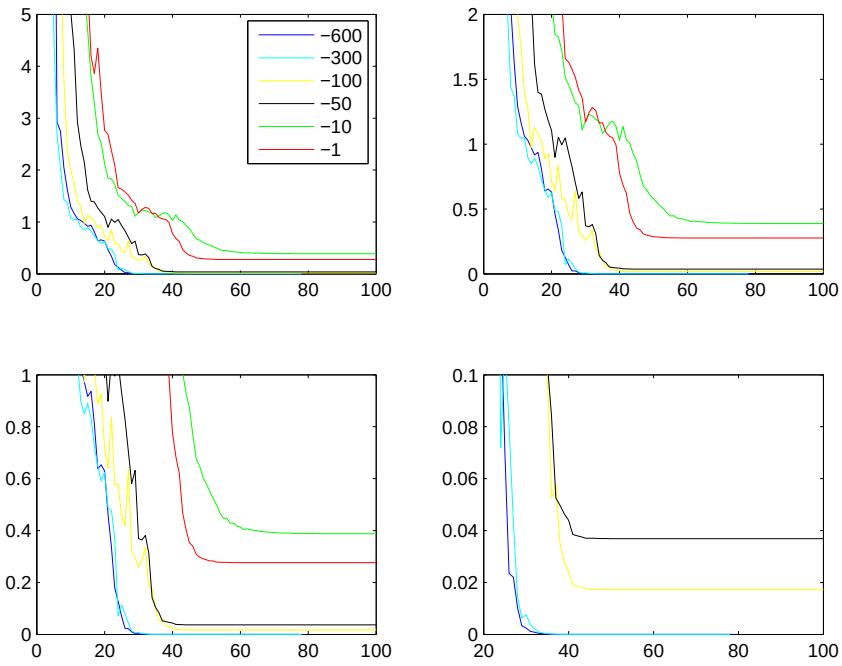

Figura 2.9: Asimetría para la función $F_{Griewangk}$ con un tamaño de población de 100 puntos

La figura 2.9 muestra el seguimiento del óptimo para cada intervalo. De la misma, se extraen algunas conclusiones importantes. Primero, no hay una diferencia considerable en el comportamiento del algoritmo cuando el extremo izquierdo está entre -600 y -300. Segundo, la generación a partir de la cual el algoritmo acelera su convergencia al óptimo (explotación del espacio de búsqueda) aumenta a medida que el intervalo se acerca a cero (óptimo de la función). Tercero, cuando el intervalo es mayor o igual a -10 el óptimo no es menor que 0.1 y para -100 y -50, el algoritmo no logra acercarse al óptimo (error de 10^{-7}). En la figura 2.10 se llegan a conclusiones similares, aunque la más importante es, que la generación a partir de la cual el algoritmo comienza a hacer la explotación, se aleja a medida que se estrecha el extremo izquierdo hacia el óptimo, es decir, dedica las generaciones iniciales a explorar el espacio de búsqueda.

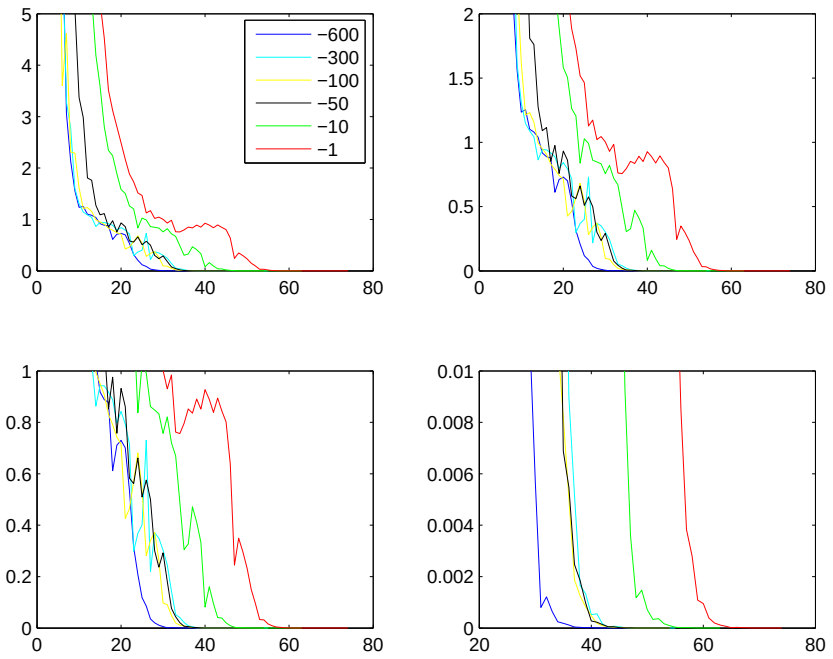

Figura 2.10: Asimetría para la función $F_{Griewangk}$ con un tamaño de población crítica óptimo en cada intervalo

2.10. Conclusiones

En este capítulo se realizó un estudio crítico sobre los métodos de aprendizaje de redes Bayesianas basados en pruebas de independencia. Como resultado se propusieron tres nuevos EDA que utilizan modelos Bayesianos generales: el $\text{CBEDA}_{\text{TPDA}}$, el $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{CMMHC}}$. Un resultado importante fue la modificación del proceso de orientación de las aristas en el algoritmo TPDA original, para que utilice optimización de métrica al igual que el MMHC. La utilización de algoritmos de aprendizaje de redes generales que utilizan modelos híbridos (*independencias + optimización de métrica*) es un resultado novedoso en el campo de los EDA. Los CBEDA rompen con el mito que existía alrededor de este tipo de métodos y su utilización en EDA. A continuación se listan los resultados alcanzados en el capítulo.

1. Análisis de los métodos de aprendizaje basados en restricciones en el dominio del aprendizaje automatizado y su aplicación al entorno de los EDA, dando lugar a una nueva clase de algoritmos llamada CBEDA y que da origen a los métodos basados en restricciones.
2. Introducción de los métodos de aprendizaje estructural de redes Bayesianas basados en restricciones como métodos para la estimación de la distribución de los EDA.
3. Análisis de los algoritmos TPDA y MMHC para su utilización como algoritmos de aprendizaje de la distribución del EDA a partir de los excelentes resultados mostrados por PADA.
4. Introducción de tres nuevos algoritmos, el $\text{CBEDA}_{\text{TPDA}}$, el $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{CMMHC}}$ basados en el aprendizaje con restricciones y optimización de métricas.

5. Diseño de las estructuras de datos, programación en lenguaje C++ y puesta a punto de los programas que implementan los CBEDA.

Con respecto a los resultados experimentales se mostró que:

1. Los algoritmos $\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{MMHC}}$ son una alternativa a los EDA existentes (EBNA, BOA). Se evidenció que resuelven problemas univariados, aunque no mejorando al UMDA, pero sí al BOA y al EBNA, con respecto al número de evaluaciones.
2. Los CBEDA detectan la estructura de la B-función en muchas menos generaciones que el EBNA.
3. El experimento de escalabilidad para RBUF demostró que el $\text{CBEDA}_{\text{MMHC}}$ tiene un comportamiento similar al $\text{BOA}\,(\text{k}=1)$ y al EBNA, aunque casi siempre realiza menos evaluaciones que ambos.
4. En cuanto a la calidad del modelo aprendido de las variantes de algoritmos CBEDA, el que mejor detecta la estructura es el poliárbol, producto a que las B-funciones utilizadas tienen esta estructura. También se demostró la necesidad de disponer de un buen método de orientación de aristas para lograr mayor porciento de éxito y menor número de evaluaciones.
5. La importancia de disponer de métodos en EDA que detectan la estructura de la función objetivo, tanto para el FDA como para algoritmos que utilizan aprendizaje como el $\text{CBEDA}_{\text{MMHC}}$.
6. El $\text{CBEDA}_{\text{MMHC}}$ y el $\text{CBEDA}_{\text{TPDA}}$ optimizan funciones con cardinalidades enteras. La comparación del $\text{CBEDA}_{\text{MMHC}}$ con EBNA demostró, que a medida que aumenta la cardinalidad del problema, el primero realiza menos evaluaciones.
7. El $\text{CBEDA}_{\text{TPDA}}$ tiene un comportamiento similar al $\text{MK}-\text{EDA}_2$ en la solución al problema de la predicción de la estructura de una proteína. Este resultado conduce a no tener que utilizar modelos complejos de aprendizaje como las redes de Markov, basta utilizar modelos Bayesianos con la ventaja adicional de no tener que restringir la estructura.
8. El $\text{CBEDA}_{\text{CMMHC}}$ obtiene resultados comparable a otros algoritmos del estado del arte en cuanto a la eficiencia evaluativa, realizando en algunos casos menor número de evaluaciones (función F_{Sphere}) que los demás algoritmos o muy cercanos a ellos (F_{Ackley} y $F_{Griewangk}$).
9. El $\text{CBEDA}_{\text{MMHC}}$ no escapa a la complejidad adicionada cuando el intervalo de definición de la función objetivo se hace asimétrico o el óptimo de la misma se encuentra cercano a uno de los extremos. En este caso el algoritmo necesita aumentar el tamaño de población para realizar una correcta estimación de la distribución de probabilidad.
10. A medida que uno de los extremos del intervalo de definición de la función se acerca al óptimo el momento en que el algoritmo comienza a realizar la *explotación* del espacio de búsqueda se extiende en el tiempo.

CAPÍTULO 3

ALGORITMOS DE ESTIMACIÓN DE DISTRIBUCIONES BASADOS EN PRUEBAS DE INDEPENDENCIA. MODELOS BAYESIANOS PARALELOS

En el capítulo 2 se presentan tres algoritmos de estimación de distribuciones basados en pruebas de independencia para dominio discreto y se extiende una de las propuestas a dominio continuo ($CBEDA_{CMMHC}$). En este capítulo se proponen tres nuevos algoritmo CBEDA paralelos: el algoritmo $pCBEDA_{LPA}$, el $pCBEDA_{MMHC}$ y el $pCBEDA_{TPDA}$ que utilizan el modelo maestro/trabajador de la programación paralela. La meta fundamental es disminuir los tiempos de cómputo sin afectar la eficiencia evaluativa de las versiones secuenciales.

3.1. Introducción

A lo largo del desarrollo de esta investigación, el trabajo se centra en el estudio de la eficiencia de los algoritmos EDA. En el capítulo anterior se presentaron algoritmos secuenciales que disminuyen el número de evaluaciones con respecto a los basados en optimización de métricas.

Uno de los objetivos fundamentales de la investigación es el salto de los resultados teóricos en EDA a las aplicaciones prácticas. Desde este punto de vista no se puede pasar por alto la reducción del tiempo de ejecución haciéndose imprescindible la utilización de técnicas paralelas para lograr este objetivo. A partir de las propuestas secuenciales eficientes evaluativamente, se dedican esfuerzos a encontrar una implementación paralela que permita cumplimentar el objetivo final de lograr un EDA eficiente en tiempo de ejecución.

Las razones que motivan la investigación del tema tratado en este capítulo se resumen de la siguiente manera:

1. La necesidad inmediata de disponer en EDA de algoritmos paralelos que permitan abordar problemas de optimización de mayor complejidad reduciendo el tiempo de ejecución.
2. Al estudiar las bondades de los algoritmos de aprendizaje de redes Bayesianas basados en restricciones, en la fase de construcción de la distribución de búsqueda del EDA, se proponen versiones paralelas de los mismos que permitan su utilización por parte de la comunidad científica.

Durante el desarrollo de este capítulo se persiguen los siguientes objetivos:

1. Diseñar e implementar tres nuevos algoritmos paralelos de aprendizaje de redes Bayesianas utilizando el modelo maestro/trabajador de la programación paralela.

2. Crear nuevas propuestas de EDA basados en los algoritmos de aprendizaje paralelos diseñados.
3. Diseñar e implementar en *C++* los algoritmos propuestos.
4. Evaluar el desempeño de los algoritmos paralelos en relación a la reducción del tiempo de ejecución.

Para cumplimentar los objetivos planteados se comienza por un análisis del método de aprendizaje utilizado por el algoritmo PADA a partir del cual se obtendrán dos implementaciones paralelas del mismo. En este análisis se estudia a nivel algorítmico para detectar las operaciones más costosas y que se ejecuten de forma independiente.

La hipótesis fundamental de este capítulo es que si se implementan de forma paralela los algoritmos secuenciales propuestos en el capítulo 2, entonces se reduce el tiempo de ejecución de los diferentes métodos. Esta hipótesis se sustenta por los resultados obtenidos en la paralelización del algoritmo PADA [85]. Por este motivo, se extienden los resultados al aprendizaje paralelo de modelos multi-conectados.

Por último se presentan un conjunto de resultados experimentales que persiguen dar una visión general del comportamiento de los métodos propuestos en este capítulo. Los resultados se enfocan hacia la reducción del tiempo de ejecución que realizan los algoritmos.

3.2. El modelo maestro/trabajador y los EDA

Los algoritmos propuestos en este capítulo utilizan el modelo maestro/trabajador de la programación paralela. En dicho modelo el proceso maestro se dedica a la ejecución del flujo principal del algoritmo y cuando arriba a una sección de código costosa comparte el procesamiento con los procesos trabajadores. Posteriormente, el maestro recolecta y/o distribuye los resultados para continuar con el flujo principal de ejecución. Este tipo de modelo es de fácil implementación en tecnologías distribuidas como los clústeres de computadoras, utilizando el paradigma de pase de mensajes (estándar MPI [42]). Esta es la motivación esencial para utilizar este modelo, las pruebas se realizan en un clúster de computadoras. Esto no limita a que los algoritmos se ejecuten sobre otras arquitecturas como los procesadores de varios núcleos.

Una propiedad importante del modelo maestro/trabajador y de las implementaciones presentadas, es que no se modifica el comportamiento del algoritmo secuencial. Dicha propiedad permite disminuir los tiempos de cómputo y posibilitar que el algoritmo implementado escale adecuadamente a problemas de mayores dimensiones.

Este modelo se utilizado ampliamente en la implementación de EDA paralelos. Referimos al lector a la sección 1.5 para una revisión detallada sobre el tema.

3.3. Paralelización de la métrica BIC

Para realizar el proceso de optimización de la métrica BIC, la orientación del esqueleto aprendido en el MMHC y el TPDA, utiliza un procedimiento escalador de colinas paralelo, similar al propuesto en [97, 95]. Este algoritmo parte de la propiedad de la métrica BIC de poderse descomponer. Es

decir, si se actualiza la estructura S con la modificación del arco $\langle j,i\rangle$ sólo será necesario recalcular $BIC(i,S,D)$.

A la hora de desarrollar el programa paralelo no se realiza ningún cambio en el algoritmo, sólo el necesario para aprovechar la ventaja que supone descomponer el cálculo de la $BIC(S,D)$ en sub-cálculos de $BIC(i,S,D)$. Esto asegura que los resultados numéricos no sufran modificaciones con respecto a la versión secuencial pero con una notable ventaja respecto al tiempo de ejecución.

Como exponentes del modelo maestro/trabajador, existe un proceso central que realiza toda la parte secuencial y reparte tareas de cómputo entre una colección de trabajadores. La parte que se ha paralelizado es el cálculo de las $BIC(i,S,D)$, ejecutados de forma independiente en los trabajadores y finalmente devolver los resultados al maestro para continuar con la ejecución secuencial. Al lector interesado en los detalles de la implementación se le recomienda los trabajos [97, 95].

3.4. Del $\text{CBEDA}_{\text{LPA}}$ al $\text{pCBEDA}_{\text{LPA}}$

Al realizar un estudio del algoritmo LPA, se concluye que los pasos que involucran los cálculos más costosos, en cuanto a complejidad algorítmica, son los cálculos comprendidos en los pasos del 3-8 y del 9-17 (complejidad de $O(n^2)$ y $O(n^3)$, respectivamente) y en los pasos del 27-32 ($O(n^3)$). También se detecta que los cálculos que se realizan en cada uno de esos ciclos son separables. Las dos propuestas que se muestran a continuación parten de la complejidad algorítmica y la separabilidad de los cálculos presentes en dichos ciclos.

3.4.1. Primera propuesta paralela: $\text{pCBEDA}_{\text{LPA}-\text{BAL}}$

La primera propuesta parte de la separación de los cálculos que se realizan de las independencias mutuas marginales y mutuas condicionales. Como se aprecia en el algoritmo 2.1, una vez que se calculan las dependencias mutuas marginales se procede a calcular las dependencias mutuas condicionales. En este caso la propuesta es que cada proceso trabajador haga el cálculo de la primera dependencia, envíe los resultados al proceso maestro y este redistribuya nuevamente el trabajo de forma equitativa. La idea esencial es que luego de que cada trabajador calcule la dependencia mutua marginal, el número de aristas resultante puede diferir entre unos trabajadores y otros. Posteriormente, se procede a distribuir nuevamente las aristas a las que se le calculará la dependencia mutua condicional, tarea que realiza el maestro.

El funcionamiento general del algoritmo es el siguiente: primero se le asigna equitativamente al proceso maestro y a cada uno de los trabajadores, las aristas a las que se le calcularán los valores de dependencia mutua marginal. Los valores son comparados con el umbral ε_0, y de ser mayores se incluyen las aristas correspondientes en la sub-lista presente en ese proceso. Concluida esta etapa, se procede a realizar un nuevo balanceo de la carga entre el maestro y los trabajadores. Cada trabajador envía al maestro la sub-lista de dependencias mutuas marginales. Luego de recibidas todas las sub-listas se procede a asignarle equitativamente al maestro y a los trabajadores las nuevas aristas para calcular los valores de dependencia mutua condicional $Dep(X_i,X_j \mid X_k)$. En cada proceso, dichos valores son comparados con el umbral ε_1, y para aquellas aristas con $Dep(X_i,X_j \mid X_k) > \varepsilon_1$

se calcula la dependencia global $Dep_g(X_i, X_j)$. Posteriormente, cada trabajador envía al maestro las aristas resultantes y los valores de dependencia global correspondientes. Con esta información el maestro ordena las aristas decrecientemente según el valor de la dependencia global y construye el esqueleto de poliárbol, insertando una a una las aristas según el orden establecido y evitando la creación de ciclos (sólo $n-1$ aristas). Luego se le asigna, equitativamente, al maestro y a cada uno de los trabajadores los sub-grafos de la forma $X_i - X_k - X_j$ para calcular $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j)$. Seguidamente, cada trabajador envía al maestro los valores calculados y para cada uno de los sub-grafos el maestro comprueba si $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j) > 0$, en ese caso crea el patrón cabeza-cabeza $X_i \rightarrow X_k \leftarrow X_j$. Finalmente el maestro orienta las aristas no orientadas en el paso anterior y se calcula la distribución de probabilidad $p(x)$. En los algoritmos 3.1 y 3.2 se muestra el pseudo-código para el proceso maestro y el proceso trabajador del pLPA_Maestro con balance de la carga.

3.4.2. Segunda propuesta paralela: $\text{pCBEDA}_{\text{LPA}-\text{UNB}}$

Esta propuesta es igual a la anterior, excepto que no implementa el balance de la carga. Después de calcular las dependencias mutuas marginales $Dep(X_i, X_j)$ cada proceso continúa con el cálculo de las dependencias mutuas condicionales $Dep(X_i, X_j \mid X_k)$. Se parte de la suposición de que el número de aristas con $Dep(X_i, X_j) > \varepsilon_0$ no difiere de un proceso a otro. Balancear la carga para el cálculo de $Dep(X_i, X_j \mid X_k)$ incrementa el tiempo de comunicación, no compensado con la reducción proveniente de la redistribución equitativa de las aristas para el calculo de $Dep(X_i, X_j \mid X_k)$ y de $Dep_g(X_i, X_j)$.

La versión no balanceada se implementa a partir de la balanceada eliminando del pseudo-código del proceso maestro (algoritmo 3.1) las líneas de la 10 a la 12 y en el proceso trabajador (algoritmo 3.2) las líneas 10 y 11.

3.5. Del $\text{CBEDA}_{\text{MMHC}}$ al $\text{pCBEDA}_{\text{MMHC}}$

Una característica importante del algoritmo MMHC (sección 2.5) es que el cálculo del conjunto de padres e hijos de cada variable es independiente y su implementación paralela es directa.

Para construir de forma paralela el conjunto $PC(X)$ de cada variable se utilizan dos estrategias. La primera consiste en hacer un balanceo dinámico, donde el maestro iterativamente envía variables a los trabajadores para computar su conjunto $PC(X)$. Esta variante tiene el inconveniente que la comunicación es intensa. La principal ventaja es que si el número de padres e hijos difiere considerablemente para cada variable, el balance dinámico mejora el uso de los recursos de cómputo disponibles. Por otro lado se puede realizar un balance estático de los cálculos que consiste en dividir la cantidad de variables del problema entre el número de procesadores disponible ($conjunto\_de\_variables = numero\_de\_variables/procesadores$). De esta forma, cada procesador computa la misma cantidad de conjuntos $PC(X)$ suponiendo que la cantidad de variables es divisible entre el número de procesadores. El inconveniente fundamental es que, a priori, no se conoce la

Algoritmo 3.1 Pseudo-código del proceso maestro para el pLPA-BAL_Maestro

procedimiento pLPA-BAL_MAESTRO(*conjunto de datos* D, *umbrales* $\varepsilon_0,\ \varepsilon_1$)
  Comenzar con un grafo sin aristas G y una lista vacía L
  Enviar D, ε_0, y ε_1 a los trabajadores
  para cada $X_i, X_j \in X \mid i \neq j$, asignados al maestro **hacer**
    Computar $Dep(X_i, X_j)$
    si $Dep(X_i, X_j) > \varepsilon_0$ **entonces**
      Adicionar el arco $\langle X_i, X_j \rangle$ a L
    fin si
  fin para
  Recibir todas las lista L_i de cada trabajador y mezclarlas en L
  Particionar L equitativamente según la cantidad de procesos
  Enviar las sub-listas a los trabajadores para calcular la dependencia mutua condicional
  para cada $\langle X_i, X_j \rangle$ en L **hacer**
    para cada $X_k \in \mathbf{X}$ **hacer**
      Computar $Dep(X_i, X_j \mid X_k)$
      si $Dep(X_i, X_j) < \varepsilon_1$ **entonces**
        Eliminar el arco $\langle X_i, X_j \rangle$ de L
        Seleccionar el próximo arco $\langle X_i, X_j \rangle$ en L
      fin si
    fin para
  fin para
  para cada $\langle X_i, X_j \rangle$ en L **hacer**
    Computar $Dep_g(X_i, X_j)$
  fin para
  Recibir todas las lista L_i y Dep_g de cada trabajador. Mezclar cada L_i en L
  Ordenar L en orden decreciente del valor de $Dep_g(X_i, X_j)$
  repetir
    Seleccionar la próxima arista $\langle X_i, X_j \rangle$ en L
    si $\langle X_i, X_j \rangle$ no crea un ciclo en G **entonces**
      Adicionar $\langle X_i, X_j \rangle$ a G
    fin si
  hasta $n-1$ aristas hayan sido adicionadas a G
  Enviar a los trabajadores los sub-grafos $X_i - X_k - X_j \in G$ para calcular $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j)$
  para cada $X_i - X_k - X_j \in G$ asignado al maestro **hacer**
    Calcular $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j)$
  fin para
  Recibir de cada trabajador $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j)$
  para cada $X_i - X_k - X_j \in G$ **hacer**
    si $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j) > 0$ **entonces**
      Orientar patrón cabeza-cabeza $X_i \rightarrow X_k \leftarrow X_j$
    fin si
  fin para
  Orientar el resto de las aristas aplicando alguna función de costo
  Computar $p(x) = \prod_{i=1}^{n} p(x_{j1(i)}, \ldots, x_{jr(i)})$
fin procedimiento

Algoritmo 3.2 Pseudo-código del proceso trabajador para el pLPA-BAL_Trabajador

1: **procedimiento** pLPA-BAL_TRABAJADOR(*conjunto de datos* D, *umbrales* $\varepsilon_0,\ \varepsilon_1$)
2: Recibir D, ε_0, y ε_1 del maestro
3: Comenzar con una lista vacía L
4: **para cada** $X_i, X_j \in X \mid i \neq j$, asignados al trabajador **hacer**
5: Computar $Dep(X_i, X_j)$
6: **si** $Dep(X_i, X_j) > \varepsilon_0$ **entonces**
7: Adicionar el arco $\langle X_i, X_j \rangle$ a L
8: **fin si**
9: **fin para**
10: Enviar la lista L al maestro
11: Recibir del maestro la sub-lista L para calcular la dependencia mutua condicional
12: **para cada** $\langle X_i, X_j \rangle$ en L **hacer**
13: **para cada** $X_k \in \mathbf{X}$ **hacer**
14: Computar $Dep(X_i, X_j \mid X_k)$
15: **si** $Dep(X_i, X_j) < \varepsilon_1$ **entonces**
16: Eliminar el arco $\langle X_i, X_j \rangle$ de L
17: Seleccionar el próximo arco $\langle X_i, X_j \rangle$ en L
18: **fin si**
19: **fin para**
20: **fin para**
21: **para cada** $\langle X_i, X_j \rangle$ en L **hacer**
22: Computar $Dep_g(X_i, X_j)$
23: **fin para**
24: Enviar la lista L y Dep_g al maestro
25: Recibir del maestro los sub-grafos $X_i - X_k - X_j \in G$ para calcular $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j)$
26: **para cada** $X_i - X_k - X_j \in G$ asignado al trabajador **hacer**
27: Calcular $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j)$
28: **fin para**
29: Enviar al maestro $Dep(X_i, X_j \mid X_k) - Dep(X_i, X_j)$
30: **fin procedimiento**

cantidad de padres e hijos que tiene un nodo, implicando diferencias en el tiempo de cómputo de los $PC(X)$ asignados a los procesadores. Tiene la ventaja que la comunicación se realiza solo una vez al finalizar el cómputo de todos los padres e hijos de cada nodo. En esta investigación se implementa la segunda variante, justificado porque en la práctica un problema de optimización no debe tener cantidades altas de dependencias entre las variables.

Algoritmo 3.3 Pseudo-código del proceso maestro para el pMMHC

```
1:  procedimiento pMMHC_MAESTRO(conjunto de datos D)
2:      Enviar los datos D a cada trabajador
3:      Computar de forma paralela las dependencias marginales
4:      para cada X_i asignada al maestro hacer
5:          PC(X_i) = MMPC(X_i, D)
6:      fin para
7:      Intercambiar el conjunto PC con todos los procesos
8:      BN = {}
9:      repetir
10:         Mediante un método escalador de colinas paralelo, aplicar los operadores
11:         DeleteArc, InvertArc y AddArc (X → Y se puede adicionar solo si
12:         Y ∈ PC(X))
13:     hasta no se puedan obtener mejoras a la BN actual
14:     retornar BN
15: fin procedimiento
```

Algoritmo 3.4 Pseudo-código del proceso trabajador para el pMMHC

```
1:  procedimiento pMMHC_TRABAJADOR
2:      Recibir los datos D del proceso maestro
3:      Computar de forma paralela las dependencias marginales
4:      para cada X_i asignada al trabajador hacer
5:          PC(X_i) = MMPC(X_i, D)
6:      fin para
7:      Intercambiar el conjunto PC con todos los procesos
8:      BN = {}
9:      repetir
10:         Mediante un método escalador de colinas paralelo, aplicar los operadores
11:         DeleteArc, InvertArc y AddArc (X → Y se puede adicionar solo si
12:         Y ∈ PC(X))
13:     hasta no se puedan obtener mejoras a la BN actual
14: fin procedimiento
```

Los algoritmos 3.3 y 3.4 muestran el pseudo-código para el proceso maestro y trabajador, respectivamente. Inicialmente, el proceso maestro envía los datos (conjunto de puntos seleccionados)

a todos los procesos, y se hace el cómputo paralelo de la dependencia marginal, de forma similar a como se realiza en el algoritmo pCBEDA$_{\text{LPA}}$ (sección 3.4). Posteriormente se hace una división de la carga basada en la cantidad de procesos que participan en la ejecución paralela. A cada proceso le corresponden $\frac{n}{p}$ variables, donde n es la cantidad de variables y p es la cantidad de procesos. Una vez calculados todos los conjuntos de padres e hijos de cada variable ($PC(X)$) se hace un intercambio de dicha información entre todos los procesos participantes y se procede a la orientación del esqueleto, utilizando un procedimiento escalador de colinas paralelo similar al propuesto en [97, 95] y descrito en la sección 3.3.

3.6. Del CBEDA$_{\text{TPDA}}$ al pCBEDA$_{\text{TPDA}}$

La implementación paralela del algoritmo CBEDA$_{\text{TPDA}}$ es más complicada que las anteriores, producto a la necesidad de mantener de forma centralizada un grafo que se construye de forma incremental. No obstante, existen dos fases bien definidas y se encuentran presentes en los algoritmos anteriores. La primera es el cómputo de las dependencias marginales y al igual que el pCBEDA$_{\text{LPA}}$ y el pCBEDA$_{\text{MHHC}}$ se realiza de forma paralela. La segunda etapa bien definida es la orientación del esqueleto al terminar las tres fases del TPDA que se realiza de forma paralela como se describe en la sección 3.3.

Los algoritmos 3.5 y 3.6 muestran el pseudo-código para el proceso maestro y trabajador, respectivamente. Inicialmente el proceso maestro envía los datos a todos los procesadores trabajadores, posteriormente y de forma paralela, se calcula la dependencia marginal y el maestro recolecta toda la información que es almacenada en una lista L ordenada según el valor de dependencia marginal. Con esta lista se construye el árbol de peso mínimo y se almacena en G, cada arista que es agregada a G se elimina de L. Posteriormente comienzan las fases de engrosado del grafo G y su refinado. En cada una de estas fases, cada vez que se necesite computar alguna dependencia condicional, el cálculo es dividido entre el maestro y los trabajadores. El mismo es de vital importancia porque el conjunto condicionante de una arista $\langle i, j \rangle$ tiene varias variables y calcular las dependencias condicionales se hace exponencial, teniendo en cuenta, que recorre todos los subconjuntos del condicionante. Por último se realiza la orientación del grafo G mediante un proceso de optimización de métrica paralelo.

Se destaca la complejidad de los cálculos y el mantenimiento del grafo centralizado y esperar que este algoritmo obtenga menores reducciones en tiempo con respecto al pCBEDA$_{\text{LPA}}$ y al pCBEDA$_{\text{MMHC}}$.

Algoritmo 3.5 Pseudo-código del proceso maestro para el pTPDA

1: **procedimiento** pTPDA_MAESTRO(*conjunto de datos D umbral* ε)
2: Enviar los datos D a cada trabajador
3: Computar de forma paralela las dependencias marginales,
4: almacenar las aristas en una lista L ordenadas descendentemente por el valor de dependencia
5: Crear un árbol G de peso mínimo utilizando el algoritmo Chow-Liu y la lista L de aristas
6: // Engrosado
7: **para cada** $\langle X_i, X_j \rangle$ en L **hacer**
8: Enviar a los trabajadores la orden de calcular las dependencias condicionales de la arista $\langle X_i, X_j \rangle$
9: Recolectar de los trabajadores el valor de dependencia condicional de $\langle X_i, X_j \rangle$
10: Agregar la arista $\langle X_i, X_j \rangle$ a G si se cumple la condición que permite agregarla
11: **fin para**
12: // Refinado
13: **para cada** $\langle X_i, X_j \rangle$ en G **hacer**
14: $G^{'} = G - \langle X_i, X_j \rangle$
15: Enviar a los trabajadores la orden de calcular las dependencias condicionales de la arista $\langle X_i, X_j \rangle$ dados sus vecinos
16: Recolectar de los trabajadores el valor de dependencia condicional de $\langle X_i, X_j \rangle$
17: $G = G^{'}$ si se cumple la condición de exclusión
18: **fin para**
19: $BN = \{\}$
20: **repetir**
21: Mediante un método escalador de colinas paralelo, aplicar los operadores
22: *DeleteArc*, *InvertArc* y *AddArc* ($X \rightarrow Y$ se puede adicionar solo si
23: Y es vecino de X
24: **hasta** no se puedan obtener mejoras a la BN actual
25: **retornar** BN
26: **fin procedimiento**

Algoritmo 3.6 Pseudo-código del proceso trabajador para el pTPDA

1: **procedimiento** pTPDA_TRABAJADOR
2: Recibir los datos D y el umbral ε del proceso maestro
3: Computar de forma paralela las dependencias marginales
4: **mientras** Orden de calcular dependencia condicional **hacer**
5: Recibir del maestro la arista $\langle X_i, X_j \rangle$ y los conjuntos condicionantes para calcular la dependencia condicional
6: Calcular dependencia condicional
7: Enviar al maestro los valores de dependencia condicional de $\langle X_i, X_j \rangle$
8: **fin mientras**
9: $BN = \{\}$
10: **repetir**
11: Mediante un método escalador de colinas paralelo, aplicar los operadores
12: *DeleteArc*, *InvertArc* y *AddArc* ($X \rightarrow Y$ se puede adicionar solo si
13: Y es vecino de X
14: **hasta** no se puedan obtener mejoras a la BN actual
15: **fin procedimiento**

3.7. Resultados experimentales

Una vez realizadas las propuestas paralelas de los algoritmos de aprendizaje del $\text{pCBEDA}_{\text{LPA}}$, $\text{pCBEDA}_{\text{MMHC}}$ y $\text{pCBEDA}_{\text{TPDA}}$ en esta sección se muestran los resultados experimentales.

La implementación de los algoritmos paralelos se realizó en ANSI C++ utilizando el estándar de comunicación MPI mediante la biblioteca de intercambio de mensajes MPICH 1.2.5. Los algoritmos se probaron en un clúster de computadoras de ocho nodos. Cada nodo está formado por un Pentium IV a 2.4 GHz, 512 MB de RAM y 512 KB de caché. Los mismos se encuentran interconectados mediante tres tipos de redes: Fast Ethernet, Gigabit Ethernet y Myrinet. El sistema operativo que utiliza cada nodo es RedHat Linux 7.2.

3.7.1. Resultados del $\text{pCBEDA}_{\text{LPA-BAL}}$ y del $\text{pCBEDA}_{\text{LPA-UNB}}$

En esta sección se analiza el comportamiento de las dos propuestas paralelas del $\text{pCBEDA}_{\text{LPA}}$ sobre tres redes de comunicación: Fast Ethernet, Gigabit Ethernet y Myrinet. Las funciones objeto de estudio fueron la F_{OneMax}, $F_{Plateau}$ y F_{Muhl}, muy utilizados en la experimentación con EDA. Primero se describen los parámetros utilizados para los experimentos, y después se analizan los resultados. Se realizan algunos experimentos preliminares para analizar cómo los parámetros afectan el comportamiento de los algoritmos. De este análisis previo, se concluye que la mejor configuración para cada problema es la mostrada en la tabla 3.1.

Parámetro	F_{OneMax}	$F_{Plateau}$	F_{Muhl}
Tamaño del problema	300	300	300
Óptimo	300	100	240
Tamaño de población	450	450	1400
Umbral de truncamiento	0.3	0.3	0.3
Elitismo	1	1	1
ε_0	0.0029	0.0029	0.0025
ε_1	0.0019	0.0019	0.0015

Tabla 3.1: Configuración de los parámetros para las dos versiones del $\text{pCBEDA}_{\text{LPA}}$

De cada combinación de algoritmo, red y problema se ejecutan 100 corridas independientes y así obtener resultados estadísticamente significativos. Para realizar una comparación justa cada algoritmo realiza el mismo esfuerzo computacional, se detiene en una generación predefinida, la 20. Esta condición de terminación permite evaluar el modelo paralelo y sus tiempos de ejecución, y se encuentra en otros trabajos [97, 95].

El $\text{pCBEDA}_{\text{LPA}}$ disminuye el tiempo de ejecución de su homólogo secuencial

La tabla 3.2 muestra los tiempos de ejecución de cada una de las propuestas paralelas en cada problema pero solo para la red Fast Ethernet. Es de inferir que para las dos restantes redes los tiempos son inferiores, pero se decide mostrar los resultados con las medidas normalizadas de speed-up y eficiencia. Se aprecia que a medida que aumenta el número de procesadores el tiempo de ejecución

de ambas propuestas disminuye, aunque siempre la variante que utiliza balanceo ($\text{pCBEDA}_{\text{LPA}-\text{BAL}}$) obtiene mejores tiempos, excepto para el caso de la función F_{Muhl} con siete y ocho procesadores.

Proc.	F_{OneMax}		$F_{Plateau}$		F_{Muhl}	
	$pCBEDA_{BAL}$	$pCBEDA_{UNB}$	$pCBEDA_{BAL}$	$pCBEDA_{UNB}$	$pCBEDA_{BAL}$	$pCBEDA_{UNB}$
1	384.27	384.27	458.48	458.48	437.16	437.16
2	186.63	200.35	221.11	239.19	239.59	249.05
3	126.45	134.68	148.46	160.26	172.64	182.41
4	95.02	104.53	111.04	124.28	137.58	142.64
5	77.18	84.17	90.45	99.83	121.20	123.92
6	66.16	71.90	76.71	86.32	106.34	110.81
7	57.24	62.97	66.62	74.19	102.33	97.49
8	52.62	56.77	61.35	67.53	100.84	88.62

Tabla 3.2: Tiempo de ejecución para el algoritmo $\text{pCBEDA}_{\text{LPA}}$ en la generación 20

La figura 3.1 muestra el speed-up (primera columna) y la eficiencia (segunda columna) para todas las redes analizadas. De los resultados se extraen varias conclusiones. Primero las redes Gigabit Ethernet y Myrinet obtienen resultados similares, y la red Fast Ethernet muestra siempre los peores resultados. Esto implica que Fast Ethernet pierde más tiempo en el intercambio de paquetes de datos que las otras redes. Esto se hace especialmente evidente cuando crece el número de procesadores, debido a que un mayor número de procesadores provoca una utilización más intensiva de la red, implicando mayor número de conflictos y produciendo una pérdida moderada de la eficiencia. Segundo, la versión balanceada del algoritmo obtiene mejores resultados que la no balanceada. Esto quiere decir, que el número de aristas resultantes del cómputo de la dependencia marginal es diferente en cada procesador, y el mecanismo de redistribuir los cálculos es beneficioso para el algoritmo. Finalmente, otra conclusión importante es que, para los problemas F_{OneMax} y $F_{Plateau}$, el algoritmo $\text{pCBEDA}_{\text{LPA}-\text{BAL}}$ es muy eficiente. Concretamente, este logra una eficiencia de 85 % con ocho procesadores y un speed-up super-lineal (eficiencia mayor del 100 %) para cualquier configuración con menos de cuatro procesadores.

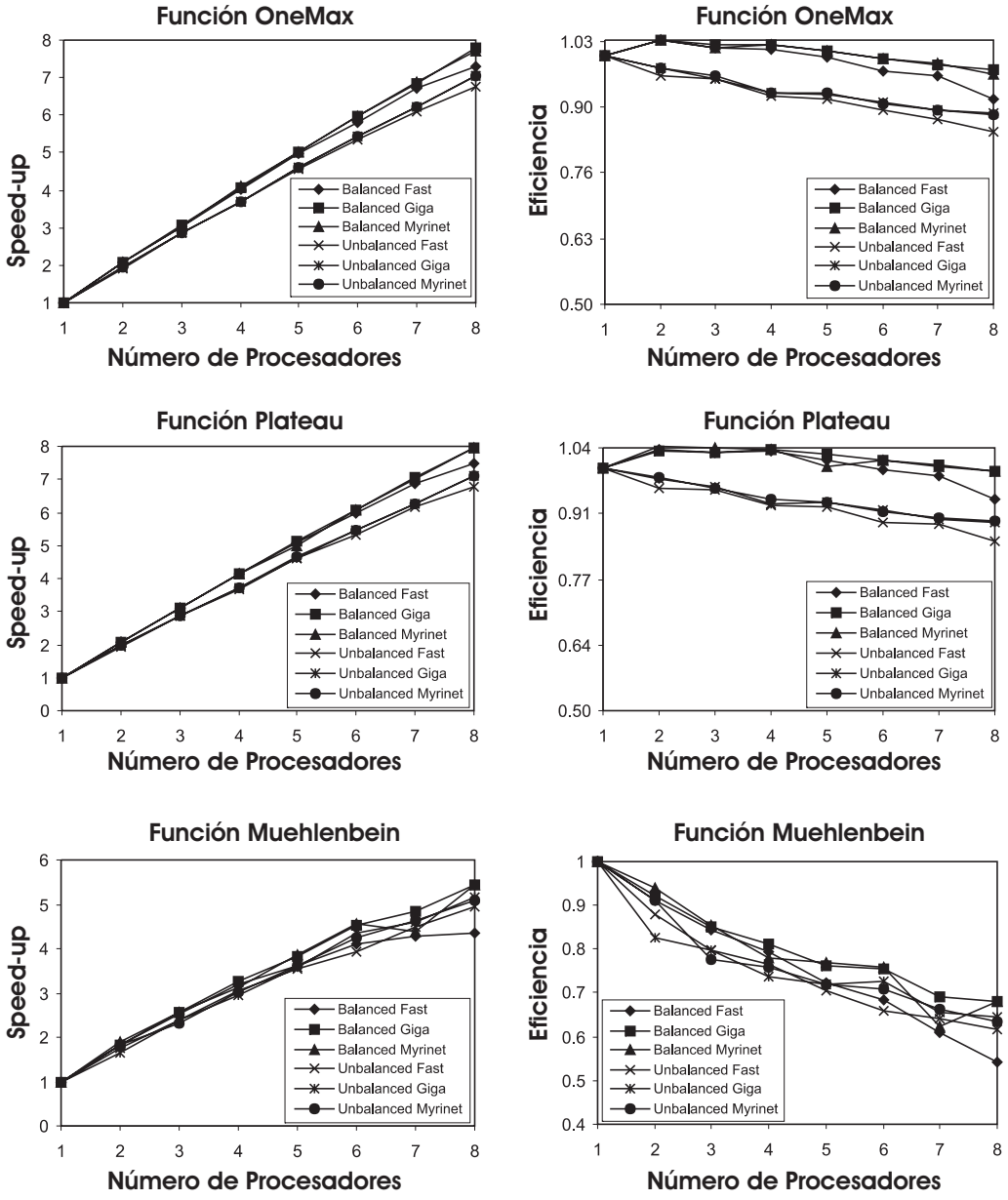

Figura 3.1: Speed-up y eficiencia para los problemas F_{OneMax}, $F_{Plateau}$, y F_{Muhl} y las dos propuestas paralelas $pCBEDA_{LPA-BAL}$ y $pCBEDA_{LPA-UNB}$

El problema F_{Muhl} es más complejo que el resto de los problemas abordados. Este necesita una población mayor para obtener resultados correctos. Al incrementarse el tamaño de la población se provoca una sobrecarga en la comunicación y consecuentemente el speed-up y la eficiencia son peores que en el resto de los problemas. Adicionalmente, este problema converge lentamente, y el método de medición no busca la solución óptima sino que la búsqueda es truncada en la generación 20. Producto de que el algoritmo se detiene en la generación 20, todos los poliárboles generados tienen un número elevado de aristas. El intercambio de aristas entre los procesadores incrementa la utilización de la red. No obstante a esta limitación, la versión paralela permite una reducción importante del tiempo de ejecución del algoritmo secuencial (de 39,38 % a 81,64 %)

Análisis del tiempo de comunicación

Los valores del tiempo de comunicación se muestran en la figura 3.2 donde se grafica el tiempo perdido por el algoritmo en comunicación y sincronización para el maestro (primera columna) y los trabajadores (segunda columna). Para todos los problemas el tiempo perdido en comunicación es pequeño, y esto implica que ambas propuestas son adecuadas para estos problemas de optimización. En el caso de las funciones F_{OneMax} y $F_{Plateau}$, la sobrecarga mayor es provocada por la configuración que utiliza Fast Ethernet con ocho procesadores. El problema F_{Muhl} es el que pierde mayor tiempo en comunicación. Como se expuso anteriormente, esto se debe a que el tamaño de la población es mayor que en el resto de los problemas, y por consecuente, la cantidad de información a intercambiar es mayor.

También se observa que la versión no balanceada produce mayores costos de comunicación que la balanceada. Esto es algo sorprendente porque se espera que la versión no balanceada haga un menor número de intercambios. La razón de este resultado es que el tiempo, que la versión balanceada pierde en el último paso de comunicación (enviando los resultados finales al maestro), es compensado con la fase de comunicación intermedia cuando la información se distribuye de forma equitativa entre los procesadores.

3.7.2. Resultados del pCBEDA$_{\text{MMHC}}$

En esta sección se analiza el comportamiento del algoritmo paralelo cuando se optimizan los problemas F_{OneMax} (ecuación A.1), $FirstPolytree3$ y $FirstPolytree5$ (ecuaciones A.8 y A.9). De la misma manera que para el pCBEDA$_{\text{LPA}}$, primero se describen los parámetros utilizados en los experimentos, y posteriormente se analizan los resultados. Se realizan algunos experimentos preliminares para analizar cómo los parámetros afectan el comportamiento del algoritmo. Del análisis previo, se concluye que la mejor configuración para cada problema es la mostrada en la tabla 3.3.

Parámetro	F_{OneMax}	$F_{Plateau}$	F_{Muhl}
Tamaño del problema	300	201	200
Óptimo	300	71.958	68.92
Tamaño de población	1000	2000	4000
Umbral de truncamiento	0.3	0.3	0.3
p-value para prueba $\mathscr{X}^2$	0.05	0.05	0.05

Tabla 3.3: Configuración de los parámetros para el pCBEDA$_{\text{MMHC}}$

Sobre cada problema se ejecutaron 30 corridas independientes del algoritmo. Para realizar una comparación justa cada algoritmo realiza el mismo esfuerzo computacional, se detiene en una generación predefinida, la 20. Solo se analizará el speed-up y la eficiencia.

La figura 3.3 resume los resultados del speed-up y eficiencia obtenidos para los tres problemas propuestos. Se aprecia un aumento del speed-up a medida que se incrementa el número de procesadores. La eficiencia se mantiene cercana o superior al 70 % para los siete procesadores utilizados y en todos

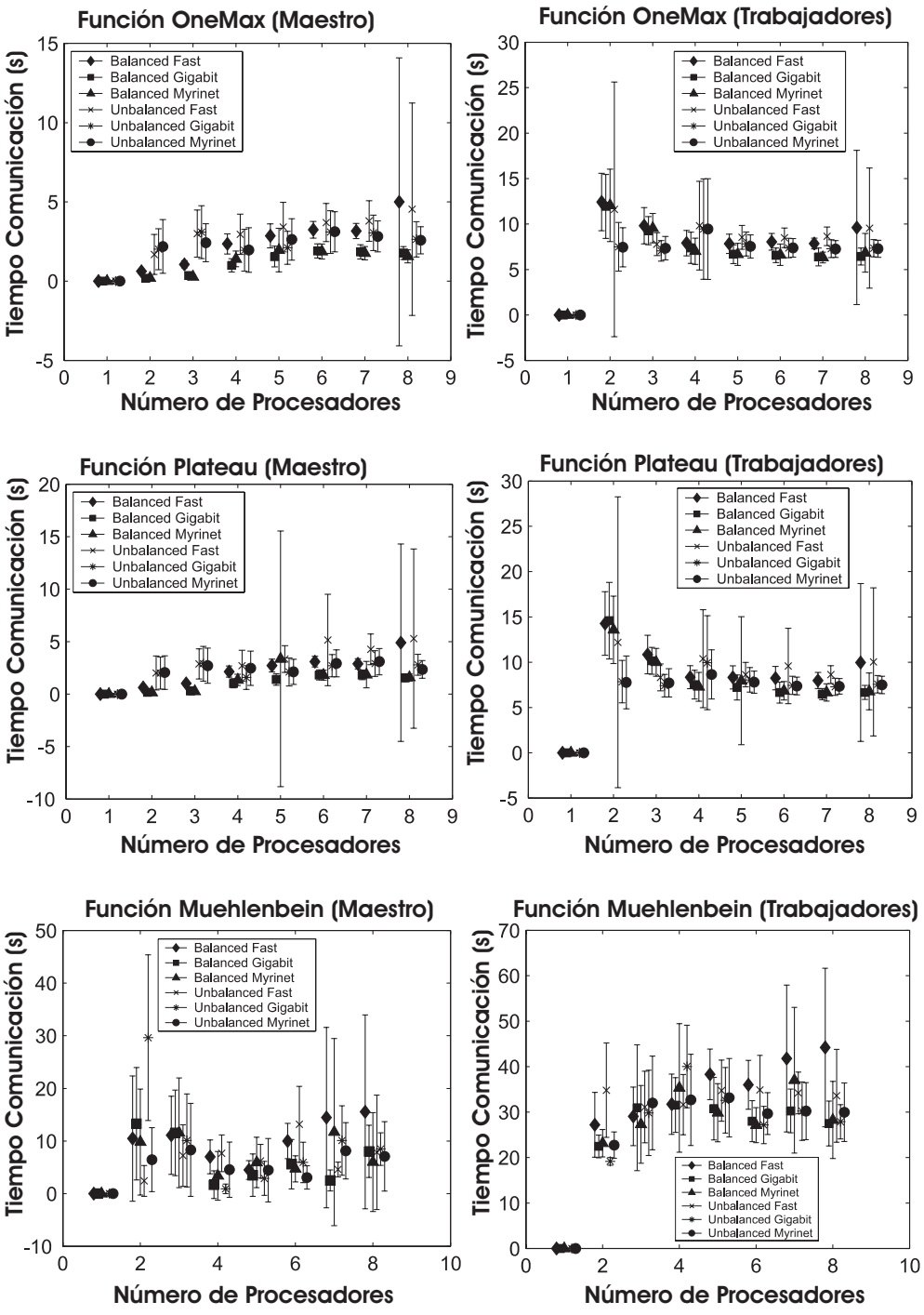

Figura 3.2: Carga en la comunicación para los problemas F_{OneMax}, $F_{Plateau}$, y F_{Muhl} y las dos propuestas paralelas $pCBEDA_{LPA-BAL}$ y $pCBEDA_{LPA-UNB}$

los problemas , lo cual indica un excelente aprovechamiento de los recursos de cómputo por parte del algoritmo.

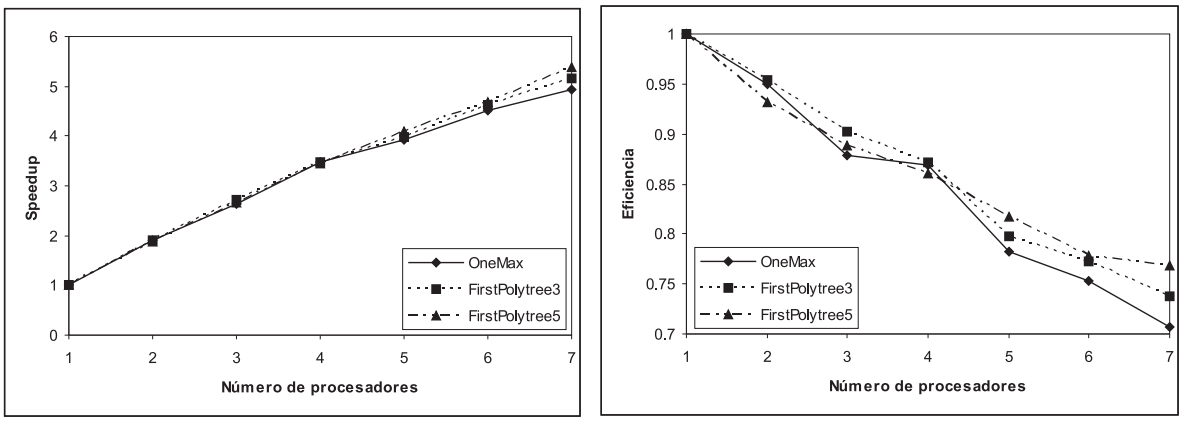

Figura 3.3: Speed-up (izquierda) y eficiencia (derecha) para el algoritmo pCBEDA$_{\text{MMHC}}$ sobre los tres problemas propuestos

Un aspecto importante a tener en cuenta es que el decrecimiento de la eficiencia y el alejamiento del speed-up del comportamiento lineal se deben al incremento de la comunicación entre el proceso maestro y los procesos trabajadores. Relacionado con esto se encuentra el tamaño de la información que intercambian los procesos, específicamente, el tamaño de la población que se utiliza para estimar la distribución de búsqueda del EDA. A medida que el problema es más complejo, se necesitan mayores tamaños de población para lograr una convergencia aceptable, que se traduce en una mayor cantidad de información a intercambiar entre los procesos. Esto queda evidenciado en los resultados obtenidos.

3.7.3. Resultados del pCBEDA$_{\text{TPDA}}$

Los resultados experimentales para el algoritmo pCBEDA$_{\text{TPDA}}$ se ajustan a los expuestos para las dos propuestas anteriores. La configuración del algoritmo y los problemas estudiados son los mismos utilizados por el pCBEDA$_{\text{MMHC}}$ (tabla 3.3). El análisis se centra en el speed-up y la eficiencia.

La figura 3.4 resume los resultados del speed-up y eficiencia obtenidos para los tres problemas propuestos. Se aprecia un aumento del speed-up a medida que se incrementa el número de procesadores. De la misma forma, la eficiencia para todos los problemas se mantiene cercana o superior al 60 % para los siete procesadores utilizados, lo cual indica un buen comportamiento del algoritmo.

En el análisis resalta que los resultados para el pCBEDA$_{\text{TPDA}}$ no son similares a los obtenidos por el pCBEDA$_{\text{LPA}}$ y el pCBEDA$_{\text{MMHC}}$, con eficiencias superiores al 70 %. Como se explicó en la sección 3.6, esto se debe en gran medida a las partes secuenciales del algoritmo no paralelizadas. En un futuro se debe analizar la introducción de la programación multi-hilos para aprovechar mejor los recursos de cómputo o hacer converger al TPDA a una versión similar a la del MMHC, sin perder su esencia.

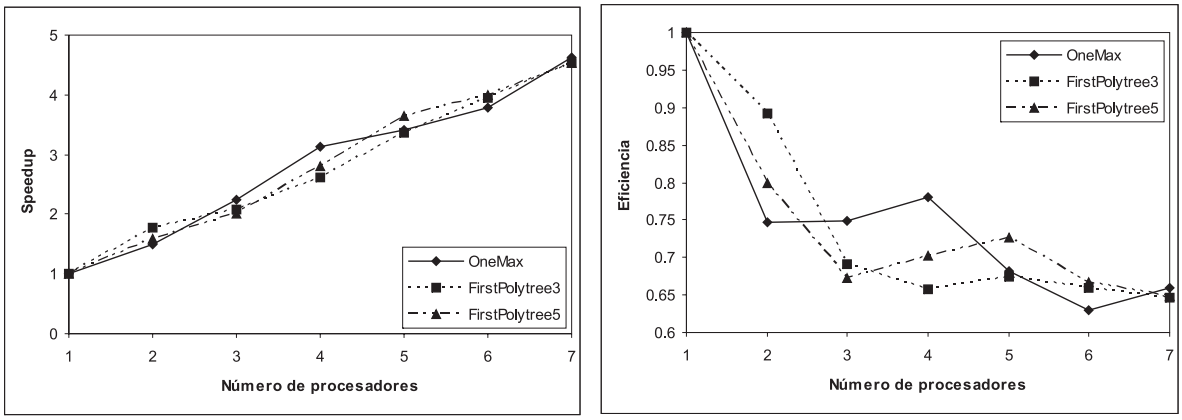

Figura 3.4: Speed-up (izquierda) y eficiencia (derecha) para el algoritmo $\text{pCBEDA}_{\text{TPDA}}$ sobre los tres problemas propuestos

3.8. Conclusiones

En este capítulo se realizó un estudio sobre los métodos de aprendizaje de redes Bayesianas basados en pruebas de independencia y su posible paralelización. Como resultado se propusieron tres nuevos algoritmos EDA paralelos que utilizan modelos Bayesianos simplemente conectados y generales: el $\text{pCBEDA}_{\text{LPA}}$, el $\text{pCBEDA}_{\text{TPDA}}$ y el $\text{pCBEDA}_{\text{MMHC}}$. Un resultado importante es la incorporación de un esquema paralelo en el proceso de orientación de las aristas utilizando optimización de métricas. La paralelización de algoritmos de aprendizaje de redes Bayesianas que utilizan modelos híbridos (*pruebas de independencia + optimización de métrica*) es un resultado novedoso en el campo de los EDA. A continuación se listan los resultados alcanzados en el capítulo.

1. Partiendo del modelo de aprendizaje que mejor resultados ofrece, PADA poliárbol caso cúbico, se propusieron e implementaron dos variantes paralelas basadas en el modelo maestro/trabajador, el $\text{pCBEDA}_{\text{LPA}-\text{BAL}}$ y el $\text{pCBEDA}_{\text{LPA}-\text{UNB}}$.
2. A partir del estudio de la propuesta del algoritmo $\text{CBEDA}_{\text{MMHC}}$ se propuso e implementó una variante paralela basada en el modelo maestro/trabajador. El algoritmo propone un balance de la carga estático en la etapa de construcción del esqueleto de la red Bayesiana.
3. A partir del estudio de la propuesta del algoritmo $\text{CBEDA}_{\text{TPDA}}$ se propuso e implementó una variante paralela basada en el modelo maestro/trabajador. El algoritmo utiliza los procesos esclavos para los cálculos de las dependencias condicionales.

Desde el punto de vista experimental se arribó a las siguientes conclusiones:

1. En general, se observa que los modelos paralelos permiten una reducción considerable del tiempo de ejecución con respecto a las versiones secuenciales, obteniendo valores de eficiencia aceptables.
2. Los algoritmos $\text{pCBEDA}_{\text{LPA}}$ muestran una adecuada escalabilidad, una cualidad promisoria para las aplicaciones reales. La versión balanceada $\text{pCBEDA}_{\text{LPA}-\text{BAL}}$ muestra mejor speed-up/eficiencia que la versión no balanceada $\text{pCBEDA}_{\text{LPA}-\text{UNB}}$.

3. De todos las propuestas, los que mejor utilización hacen de las tecnologías paralelas son el $pCBEDA_{LPA}$ y el $pCBEDA_{MMHC}$, siendo el de peor resultados el $pCBEDA_{TPDA}$, aunque esto no descarta su utilización en el campo de la optimización evolutiva paralela y más aún con los excelentes resultados mostrados en el capítulo 2 por su homólogo secuencial.

CAPÍTULO 4

ALGORITMOS DE ESTIMACIÓN DE DISTRIBUCIONES DESCENTRALIZADOS

En el capítulo 1 se presentan los conceptos y algoritmos que se utilizan en la tesis, así como una detallada revisión sobre los algoritmos EDA y los niveles de paralelismo presentes en los mismos. En este capítulo se diseñan dos modelos descentralizados de EDA: el modelo de islas y el modelo celular. La meta fundamental es mejorar la eficiencia evaluativa de la versión centralizada en términos del número de evaluaciones, además de disminuir el tiempo total de ejecución en el modelo de islas.

4.1. Introducción

De la misma manera que en otras áreas del aprendizaje y la optimización, reducir el costo del proceso de búsqueda de la solución es una cuestión crítica en EDA. Este costo usualmente se cuantifica como el número de evaluaciones de la función objetivo realizadas por el algoritmo. Pero, reducir el tiempo total de ejecución del algoritmo también es un punto importante en aplicaciones reales, en las cuales algunas operaciones llevan al algoritmo a tener tiempos de cómputo prohibitivos.

Este capítulo trata de dar respuesta a una aparente cuestión simple: ¿Cómo se puede reducir el número total de evaluaciones realizadas por un EDA en contraste con su tiempo de ejecución? Algunas de las ideas incluyen el uso de técnicas híbridas, combinando la búsqueda global con búsqueda local. La descentralización (distribuidos o celulares) también alivian el problema del esfuerzo evaluativo [8] y su paralelización permite disminuir el tiempo de ejecución.

Las motivaciones que conllevan al estudio de la descentralización y su aplicación en EDA son las siguientes:

1. El éxito que tiene la aplicación de la descentralización en otros EA, como los GA distribuidos y los GA celulares, principalmente en la disminución de la cantidad de evaluaciones de la función objetivo [7, 25].
2. Fácil implementación de los algoritmos evolutivos descentralizados en arquitecturas distribuidas como los clústeres de computadoras (COW).
3. Excelente comportamiento del UMDA en muchos problemas con interacciones tanto lineales como no lineales. Esto condujo a evaluar la posibilidad de descentralizar la población para mejorar las aptitudes del UMDA y por extensión de otros EDA.
4. La ausencia de propuestas descentralizadas en el campo de los EDA tanto para dominio discreto como para continuo. Casi todas las propuestas de algoritmos descentralizados en el campo de

los algoritmos evolutivos están basados en representaciones binarias, limitando su aplicación a problemas prácticos reales.

Los objetivos que persigue este capítulo son los siguientes:

1. Mejorar las cualidades del algoritmo UMDA en relación con el número total de evaluaciones realizadas a la función objetivo.
2. Proponer y evaluar empíricamente los algoritmo dEDA y cEDA utilizando como base al UMDA, tanto para dominio discreto como continuo.
3. Diseñar e implementar un programa (C++) que materialice el estudio sobre los esquemas dEDA y cEDA, que permita experimentar y mostrar las ventajas de los esquemas descentralizados.
4. Crear una aplicación ViPoC [158, 122] con los modelos descentralizados para hacer efectiva la popularización de estos programas entre la comunidad científica que necesita de las aplicaciones pero que no conoce las técnicas paralelas y distribuidas.

La presente investigación sobre los esquemas descentralizados y su aplicación a los EDA se divide en tres partes fundamentales. Primero se hace un estudio de las técnicas distribuidas y de los parámetros que las gobiernan. A partir de este estudio se hace la primera propuesta algorítmica denominada dEDA. A continuación se estudia el modelo celular y su extensión al campo de los EDA. Por último, se presenta un estudio experimental de las dos propuestas, mostrándose la superioridad de las versiones descentralizadas.

4.2. Algoritmos descentralizados

Los EDA pueden ser paralelizados a diferentes niveles (sección 1.5), desde el aprendizaje, la simulación hasta la descentralización de la población. Este capítulo enfatiza la posibilidad de paralelizar el EDA a nivel de población, estructurando los individuos de forma tal, que colaboren en un ambiente distribuido.

Dentro de las formas más frecuente de descentralización de algoritmos evolutivos se encuentran los distribuidos (grano-grueso o modelo de islas) y los celulares (grano-fino) [6].

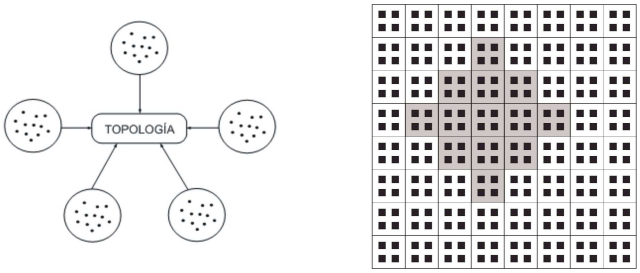

Figura 4.1: Modelos para estructurar la población en algoritmos descentralizados. A la izquierda, el modelo de islas y a la derecha, el modelo celular

El modelo computacional de grano-grueso (esquema de la izquierda en la figura 4.1) en EA se estudia profundamente por parte de la comunidad científica del tema, así como en otras ramas

de la optimización y el aprendizaje. El modelo de islas [30] está formado por sub-algoritmos separados geográficamente, cada uno con su propia sub-población. Estos sub-algoritmos pueden o no intercambiar información cada cierto tiempo, por ejemplo, permitiendo que algunos individuos migren de una isla a otra. La idea principal de este método es reinyectar periódicamente diversidad, al igual que la mutación, con la diferencia de que los individuos inyectados tienen alta probabilidad de ser buenos y así evitar una convergencia prematura de los sub-algoritmos. Diferentes islas tienden a explorar diferentes porciones del espacio de búsqueda de forma paralela, y buscar distintas soluciones para el mismo problema [165]. Dentro de cada sub-población, se ejecuta un EA secuencial estándar entre cada fase de migración.

Los algoritmos pertenecientes al modelo de grano-grueso deben ser correctamente ajustados, pues son controlados por varios parámetros que afectan su precisión y eficiencia. Dentro de los parámetros fundamentales a ajustar se encuentran: el tamaño y número de sub-poblaciones, la topología de conexión entre estas, el número de emigrantes, frecuencia entre migraciones sucesivas y el criterio para seleccionar los emigrantes y los individuos que se reemplazan con los nuevos que arriban. La importancia de estos parámetros, en la calidad de la búsqueda y en la eficiencia del algoritmo, se ha estudiado con profundidad [8, 54, 159], aunque los valores óptimos dependen del problema a solucionar.

En esta investigación se utiliza una topología de anillo unidireccional, puesto que es muy fácil de implementar y analizar (una discusión detallada de esta y otras topologías se realizan por Cantú-Paz [25]).

Los EA celulares (esquema de la derecha en la figura 4.1) se basan en el concepto de vecindario con una estructuración de la población en forma de malla. Cada rejilla almacena una pequeña cantidad de individuos (de forma general un individuo, no así en el caso de los EDA distribuidos) y al unirse con otras rejillas que se encuentran relativamente cerca forman lo que se conoce como vecindario. Existe cierto solapamiento entre los diferentes vecindarios que permiten la exploración del algoritmo y la difusión de las soluciones a otras posiciones de la malla. Dentro del vecindario es donde se lleva a cabo la explotación de las soluciones.

Otra propuesta interesante pero que queda fuera del alcance de esta tesis es la hibridación de estos modelos. De forma general la población se estructura en dos niveles, un nivel alto donde se tiene un esquema de islas y un nivel bajo donde cada una de las islas tiene la población estructurada según el esquema celular.

4.3. El modelo de islas en EDA

El modelo de islas está compuesto por diferentes sub-poblaciones para las cuales se ejecuta un algoritmo evolutivo, ya sea secuencial o distribuido. En el caso de los dEDA cada isla ejecuta un EDA, que puede ser el mismo en todas o diferentes. Otro punto importante es que cada cierto intervalo de tiempo las islas intercambian información. A partir de estas cuestiones se hace necesario definir la política de migración que define la forma en que se intercambiará dicha información.

4.3.1. Política de migración en algoritmos distribuidos

El principio de funcionamiento de un algoritmo evolutivo distribuido incluye una fase de comunicación que es gobernada por una política de migración. Esta política define cómo se desarrolla el proceso de comunicación entre las islas que componen el algoritmo y está compuesta por cinco parámetros:

- **Número de emigrantes** (m). Este es el número de individuos a intercambiar entre las islas, $m \in \{0, 1, 2 \ldots\}$. El valor 0 provoca que no exista comunicación entre las sub-poblaciones (búsqueda desconectada). Alternativamente, este parámetro puede ser medido como en porciento del tamaño de la población o como una razón.
- **Frecuencia de migración** (r). Número de generaciones en aislamiento, $r \in \{0, 1, 2 \ldots\}$. Alternativamente, este parámetro se puede medir como el número de evaluaciones de la función objetivo antes del proceso de migración. Equivalentemente, se puede medir en función del número de generaciones antes de la migración, apropiado para algoritmos generacionales.
- **Política para seleccionar los emigrantes** (S). La selección de los emigrantes se puede hacer al aplicar cualquiera de los operadores de selección disponibles en la literatura (proporcional a la aptitud, torneos, truncamiento, etc.), ejemplo, *S={mejor, aleatorio}*. Los más utilizados son el truncamiento (se seleccionan los mejores) y la selección aleatoria.
- **Política de reemplazo** (R). Este se utiliza para incorporar a la sub-población los individuos que llegan producto de la migración en la sub-población objetivo, ejemplo, *R={peores, aleatorio}*. Este parámetro decide qué individuos se reemplazan por los emigrantes que arriban a la isla.
- **Sincronización**. Este parámetro es una bandera que indica si las islas intercambian información de envío/recibo bloqueante, o si los individuos son incorporados en el momento que arriben, en cualquier momento de la búsqueda. Este parámetro clasifica el algoritmo en sincrónico o asincrónico, según el tipo de bloqueo.

En la práctica es posible combinar estos parámetros de varias formas. El algoritmo propuesto en este trabajo se puede probar con cualquier combinación de parámetros, aunque para los experimentos se mantienen algunos de ellos fijos.

4.3.2. Un algoritmo EDA basado en el modelo de islas

Luego de discutir los diferentes parámetros que afectan la paralelización de los EA de grano-grueso. Como se apuntó anteriormente, el enfoque distribuido se utiliza producto a que permite la utilización y explotación de los conglomerados de computadoras, siendo una de las plataformas paralelas más populares y disponible en laboratorios y departamentos. El EDA resultante de aplicar el modelo de islas se muestra en el algoritmo 4.1.

El algoritmo dEDA (a)sincrónico se puede ver como la combinación de *d-islas* que ejecutan un algoritmo EDA cada una. En este trabajo se hace énfasis en la utilización de los dEDA asincrónicos

Algoritmo 4.1 dEDA simple ejecutándose en cada isla

Poner $t \leftarrow 1$
Generar $N \gg 0$ puntos de forma aleatoria
mientras no se cumpla el criterio de parada **hacer**
  Evaluar la población en la función $f(x)$
  De acuerdo a un método de selección, construir un conjunto CS de M puntos
  Estimar la distribución del conjunto seleccionado $p^{CS} = p(x,t)$ a partir de **CS**
  Generar N nuevos puntos a partir de $p(x,t+1) \approx p^{CS}(x,t)$
  Enviar y recibir individuos (a)sincrónicamente de acuerdo a los parámetros de migración
  Poner $t \leftarrow t+1$
fin mientras

producto a que se ejecutan más rápido que los sincrónicos cuando las islas ejecutan el mismo tipo de algoritmo en procesadores similares [9].

La idea principal de la propuesta de algoritmo asincrónico es ejecutar en cada isla un algoritmo EDA, y periódicamente (ejemplo, después de la generación de los nuevos individuos) verificar si ha llegado el momento de efectuar la migración. En tal caso, se procede al intercambio de individuos con las islas vecinas según la topología de conexión seleccionada y al resto de los parámetros de migración. Los individuos que arriban a las islas reemplazan a los individuos seleccionados. Se seleccionan los mejores individuos para migrar y reemplazan a los peores individuos de las islas vecinas. Se espera que esta política de intercambio induzca una presión selectiva elevada que acelera la convergencia del algoritmo como un todo [24].

En una concepción más general, cada isla en un algoritmo dEDA ejecuta un EDA diferente, teniendo como resultado un dEDA *heterogéneo*, lo que constituye también una interesante línea de investigación abierta. Por ejemplo, se puede tener un dEDA de cuatro islas donde la primera ejecuta un algoritmo UMDA; la segunda, MIMIC; la tercera, EBNA; y la última, PADA. Cada algoritmo (dependiendo del problema) tiene ventajas potenciales y debilidades que se conjugan con los rasgos de los otros algoritmos participantes. Para tratar con las diferencias en el tiempo de ejecución de cada algoritmo, se sugiere que el algoritmo distribuido se implemente asincrónicamente, para explotar eficientemente la potencia de cada componente. En un escenario heterogéneo diferente, cada isla ejecuta el mismo algoritmo pero con parámetros diferentes.

En esta tesis, la implementación se basa en la ejecución de un dUMDA asincrónico (para dominio discreto y dominio continuo) en cada isla, es decir, se utilza un dEDA homogéneo. El pseudo-código del dUMDA se muestra en el algoritmo 4.2.

4.4. El modelo celular en EDA

Los algoritmos evolutivos celulares (cEA) están dotados de una estructura espacial interna que permite la diversidad de la aptitud y el fenotipo de los individuos mediante un número de iteraciones [153] a diferencia de los algoritmos centralizados. Adicionalmente, algunos trabajos han establecido las

Algoritmo 4.2 dUMDA, ejecutando en cada isla el UMDA

1: Poner $t \leftarrow 1$
2: Generar $N \gg 0$ puntos de forma aleatoria
3: **mientras** no se cumpla el criterio de parada **hacer**
4: Evaluar la población en la función $f(x)$
5: De acuerdo a un método de selección, construir un conjunto CS de M puntos
// Estimar la distribución del conjunto seleccionado $p^{CS} = p(x,t)$ a partir de CS
6: **si** dominio == discreto **entonces**
7: $p^{CS}(x,t) = \prod_{i=1}^{n} p^{CS}(x_i,t)$
8: **sino**
9: $p^{CS}(x,t) = f(x,t,\mu^t,\sigma^t) = \prod_{i=1}^{n} f(x_i,t,\hat{\mu}_i^t,\hat{\sigma}_i^t)$
10: $\hat{\mu}_i^t = \overline{X}_i^t = \frac{1}{N}\sum_{r=1}^{N} x_{i,r}^t$
11: $\hat{\sigma}_i^t = \sqrt{\frac{1}{N}\sum_{r=1}^{N}(x_{i,r}^t - \overline{X}_i^t)^2}$
12: **fin si**
13: Generar N nuevos puntos a partir de $p(x,t+1) \approx p^{CS}(x,t)$
14: Enviar y recibir individuos (a)sincrónicamente de acuerdo a los parámetros de migración
15: Poner $t \leftarrow t+1$
16: **fin mientras**

ventajas de utilizar cEA sobre otros modelos evolutivos para abordar problemas de optimización complejos, donde se necesita alta eficacia y un pequeño número de pasos [106, 11].

El modelo celular en EDA se introduce por Madera y col. [86] como una versión descentralizada y generalizada de los modelos celulares desarrollados para otros algoritmos evolutivos [92, 11]. En un algoritmo cEDA la población es descentralizada particionándola en pequeñas sub-poblaciones (llamadas celdas o algoritmos miembros), ordenadas en una malla toroidal e interactuando solo con las sub-poblaciones vecinas.

La organización de un cEDA se basa en una estructura tradicional de dos dimensiones de vecindades solapadas. Esta estructura se entiende mejor en términos de dos rejillas, una formada por cadenas o individuos y otra formada por conjuntos disjuntos de cadenas o individuos (sub-poblaciones). La figura 4.2 muestra una población global de 18×18 individuos (cuadrados pequeños) particionada en una rejilla toroidal de celdas (cuadrados grandes) conteniendo cuatro individuos cada una. El vecindario utilizado es el bien conocido C13, que se compone por una sub-población central y las 12 celdas más cercanas a ella (medidas con la distancia de Manhattan). Se adopta la notación utilizada po Madera y col. [86] para describir la forma (estructuración) de la población. Por ejemplo, la rejilla de la figura 4.2 se etiqueta como $2 \times 2 - 9 \times 9$.

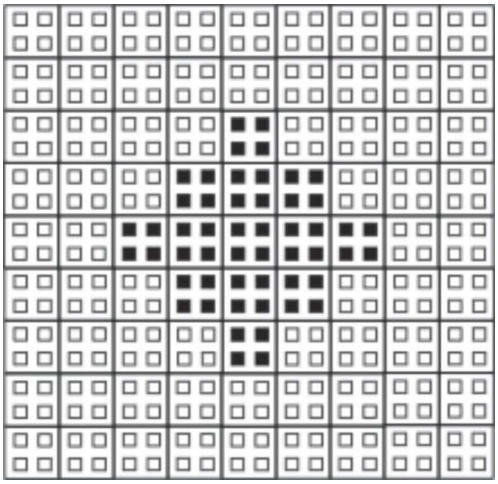

Figura 4.2: Un *cEDA* con un vecindario C13 y una estructuración de la población $2 \times 2 - 9 \times 9$

En el algoritmo 4.3 se presenta el pseudo-código del modelo de cEDA propuesto. Cada iteración del cEDA consiste de exactamente una iteración de todos los algoritmos miembros. Cada uno de estos algoritmos es responsable de actualizar exactamente una sub-población, y esto se realiza aplicando un modelo de EDA clásico local a la población compuesta por sus individuos más los individuos de las sub-poblaciones vecinas (según el vecindario definido) . En la implementación del cEDA propuesto en este capítulo, las sub-poblaciones son reemplazadas todas a la vez. Los nuevos individuos generados por los pasos de aprendizaje y muestreo local se colocan en una población temporal. Esta política de actualización es conocida como sincrónica, debido a que todos los individuos son actualizados a la misma vez en la población principal. Una alternativa a este tipo de actualización es la conocida como asincrónica, y consiste en colocar los individuos generados directamente en la población actual, siguiendo algunas reglas [48] en vez de actualizar todos los individuos simultáneamente.

Algoritmo 4.3 Pseudo-código de un cEDA simple

1: Poner $t \leftarrow 1$
2: Generar $N \gg 0$ puntos de forma aleatoria
3: **mientras** no se cumpla el criterio de parada **hacer**
4: Evaluar la población en la función $f(x)$
5: **para cada** celda **hacer**
6: De acuerdo a un método de selección, construir un conjunto local CS de M puntos
7: Estimar la distribución del conjunto seleccionado $p^{CS} = p(x,t)$ a partir de CS
8: Generar $SizeOf(celda)$ nuevos puntos a partir de $p(x,t+1) \approx p^{CS}(x,t)$
9: Insertar los individuos generados en la misma celda de una población auxiliar
10: **fin para**
11: Reemplazar la población actual por la auxiliar
12: Computar y actualizar las estadísticas
13: Poner $t \leftarrow t+1$
14: **fin mientras**

En el paso de reemplazamiento (línea 11), la vieja población puede ser tomada en cuenta (reemplazar un individuo si el nuevo que se generó es mejor) o no (reemplazar siempre los individuos de la vieja

población por los nuevos generados). La primera alternativa (llamada elitista) es preferida en esta investigación. Finalmente, el cómputo de las estadísticas (línea 12) no se encuentra comúnmente en otras propuestas algorítmicas. Sin embargo, esto se puede utilizar para monitorear la ejecución del algoritmo y tomar decisiones sobre cambios a la búsqueda adaptativa, siempre y cuando se necesite.

Una cuestión crítica en un cEDA es la computación del modelo probabilístico debido a la alta complejidad computacional que usualmente supone este paso. El lector es referido al trabajo de Madera y col. [86] para una explicación de las posibles alternativas en los esquemas de aprendizaje para cEDA. Nótese que cuando se utiliza el UMDA como algoritmo miembro del cEDA, implica que no se necesita aprender la estructura del modelo (esta es conocida), solo realizar los conteos de las frecuencias univariadas.

Hay que destacar que los experimentos evalúan las propiedades numéricas del modelo propuesto. En principio, no es necesario utilizar implementación paralela de los mismos, propiedad interesante del modelo y que lo hace flexible para su ejecución en cualquier plataforma. Una implementación paralela que mantiene el mismo comportamiento sincrónico del algoritmo es posible teniendo como única ventaja la disminución del tiempo de cómputo. También se parte de la motivación de estudiar primeramente las propiedades numéricas del modelo propuesto y luego plantear alternativas de su paralelización. En el caso de los modelos (a)sincrónicos el paralelismo sí puede traer cambios en el comportamiento numérico del algoritmo.

4.5. Resultados experimentales

En esta sección, se reportan y discuten los resultados computacionales de la aplicación de los dos modelos descentralizados propuestos en esta tesis. El conjunto de problemas seleccionados se encuentran definidos en el anexo A.

4.5.1. Resultados del modelo de islas

En este apartado se muestran los resultados de la aplicación de las dos propuestas distribuidas (discreta y continua), explicadas en la sección 4.3, sobre un conjunto de problemas de optimización. En el caso del dominio discreto se comparan el UMDAD (UMDA discreto) con dUMDAD. En el dominio continuo, compararemos UMDAC (UMDA continuo) con dUMDAC.

La metodología que se siguió para realizar los experimentos consiste en buscar el tamaño de población crítica para el cual el algoritmo UMDA resuelve los problemas propuestos con al menos un 95 % de éxito (encontrar el óptimo para el problema). El objetivo es poder realizar comparaciones del número de éxitos y del número de evaluaciones de la función objetivo. En el estudio se analizan diferentes parametrizaciones del dUMDA para estudiar la influencia de la migración en los resultados. La meta es buscar la existencia de un dUMDA que pueda realizar un número menor de evaluaciones de la función objetivo que la versión UMDA en panmixia (una sola población). Posteriormente, se reporta el estudio del tiempo total de ejecución del algoritmo para demostrar la eficiencia en tiempo.

Inicialmente se estudia el comportamiento del algoritmo para el problema F_{OneMax} y se analiza la hipótesis de que dUMDAD es más eficiente que un algoritmo UMDAD con una sola población para el resto de los problemas propuestos. No se introduce ninguna configuración especial para los algoritmos, y los parámetros se ajustan finamente para las versiones descentralizadas.

Ambos algoritmos (discreto y continuo) utilizan selección por truncamiento con un umbral de 0.3 (el mejor 30 % de la población es seleccionada). No se utiliza elitismo, la población generada reemplaza totalmente a la anterior. Para cada problema el tamaño de la población N se muestra en los cuadros de resultados. El criterio de parada es encontrar el óptimo del problema (se considera como éxito) o alcanzar 10^5 evaluaciones. El número de evaluaciones es el promedio de 100 corridas independientes del algoritmo. Para las versiones distribuidas, el número de evaluaciones total es la suma de las evaluaciones realizadas por cada isla. Todos los resultados (evaluaciones de la función objetivo y la ganancia en velocidad) son el promedio sobre las corridas donde se obtuvo éxito. En el caso continuo, la primera generación es creada utilizando una distribución normal a partir de la estimación de μ y σ en el intervalo de definición (a,b) de las variables del problema, $\mu = \frac{a+b}{2}$ y $\sigma = \frac{b-a}{6}$. Esta definición permite que el 99 % de los puntos generados se encuentran en el intervalo.

dUMDAD reduce el esfuerzo evaluativo

La sección experimental sobre dEDA comienza tratando de responder la siguiente pregunta: ¿Puede la implementación dUMDAD superar la eficiencia evaluativa de UMDAD? Empíricamente se explora la respuesta a esta pregunta al analizar el comportamiento de estos algoritmos en un conjunto de problemas.

Primero se analizan los resultados relacionados con el problema F_{OneMax}. La tabla 4.1 muestra el porciento de éxito (óptimo encontrado) y el número de evaluaciones para el UMDAD con cuatro tamaños de población distintos. En los experimentos, la función F_{OneMax} está definida sobre un vector de 1000 variables. La ejecución del algoritmo se detiene si no se encuentra el óptimo (un valor de 1000) luego de realizar 40000 evaluaciones de la función objetivo.

N	Porciento de éxito	Número de Evaluaciones
400	97 %	16437.11 ± 224.22
200	3 %	8600 ± 200
100	0 %	-
50	0 %	-

Tabla 4.1: Porciento de éxito y número de evaluaciones (media más desviación estándar) obtenido con UMDAD para diferentes tamaños de población (N) resolviendo el problema F_{OneMax} con 1000 variables

Note en la tabla 4.1 que UMDAD solo converge a la solución óptima con un porciento de éxito superior al 95 % para un tamaño de población de 400. El objetivo es reducir el costo alcanzado por el UMDAD con 400 individuos (97 % de éxito), utilizando el dUMDAD de dos y cuatro islas con tamaños de poblaciones de 200, 100 y 50 individuos, respectivamente.

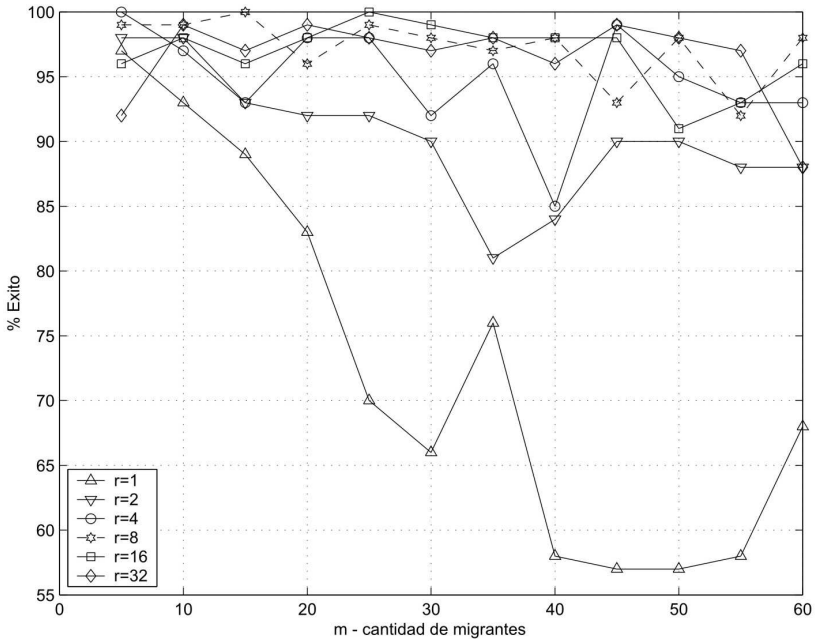

Figura 4.3: Porciento de éxito para diferentes valores de la frecuencia de migración (r) y la cantidad de emigrantes (m)

La figura 4.3 muestra el porciento de éxito para diferentes valores de los parámetros en una topología de anillo de dos islas para el problema F_{OneMax}. Se grafica una línea por cada valor de r, donde r es la frecuencia de migración. El número de emigrantes m varía de 5 a 60, que es aproximadamente el tamaño de la población seleccionada. Como se observa, el porciento de éxito se mantiene relativamente alto para la mayoría de las combinaciones de los parámetros de migración. Con la excepción de $r = 1$ (acoplamiento alto), todos los demás se mantienen superior al 90 % de éxito para todos los valores de m, probados entre cinco y 30. Para $m \leq 20$ prácticamente todos los valores de r sobrepasan el 95 % con la excepción de $r = 1$ y $r = 2$. Todo esto significa que resultados certeros se obtienen espaciando los tiempo de aislamiento (r) y/o aumentando la cantidad de individuos a intercambiar (con las excepciones mencionadas anteriormente de los sub-algoritmos con acoplamiento alto teniendo $r = 1$ y $r = 2$). Esto confirma resultados similares con otros EA reportados en el pasado [8].

En el caso de cuatro islas (figura 4.2), se concluye que la mejor configuración de (m, r) es un dUMDAD, utilizando $r = 1$ y $m = 5$. A esta conclusión se llega si se analizan todas las combinaciones donde el éxito es superior al 95 % y se selecciona la que ejecuta un menor número de evaluaciones. Sin embargo, está claro que el algoritmo es sensible a la parametrización utilizada cuando $r = 1$.

	$r=1$	$r=2$	$r=4$	$r=8$
$m=1$	99 %	99 %	98 %	97 %
	16921.21 ± 293.91	18371.43 ± 411.18	20639.18 ± 944.56	25266.67 ± 2263.24
$m=5$	95 %	99 %	97 %	96 %
	14265.26 ± 264.88	15653.06 ± 267.17	16900.00 ± 338.42	18160.00 ± 486.06
$m=10$	76 %	95 %	95 %	97 %
	13236.84 ± 237.12	14863.83 ± 264.31	16234.04 ± 283.81	17316.67 ± 332.98
$m=15$	62 %	77 %	90 %	97 %
	12954.84 ± 209.35	14736.00 ± 271.45	16094.38 ± 225.81	17204.17 ± 335.89
$m=20$	58 %	81 %	94 %	97 %
	12979.31 ± 261.41	14680.00 ± 214.89	16038.71 ± 257.51	17091.67 ± 340.79
$m=25$	53 %	80 %	89 %	93 %
	12988.68 ± 230.10	14673.42 ± 252.53	16050.00 ± 269.95	17191.30 ± 355.69
$m=30$	51 %	81 %	92 %	89 %
	12933.33 ± 206.56	14670.00 ± 244.64	16128.89 ± 298.78	17290.91 ± 362.51

Tabla 4.2: Porciento de éxito y número de evaluaciones (media más desviación estándar) obtenido para dUMDAD con cuatro islas resolviendo el problema F_{OneMax}. Los resultados se muestran para diferentes combinaciones del número de emigrantes (m) y la frecuencia de migración (r)

Resultados para otros problemas discretos

Los resultados de la tabla 4.3 permiten concluir la alta eficiencia del algoritmo dUMDAD. Nótese en la sexta columna de la tabla, que todos los resultados son estadísticamente significativos para la prueba t-student (el p-value es muy inferior a 0.05, para un nivel de significación del 95 %). Este algoritmo es más eficiente que el UMDAD para la misma razón de éxito (esto es real para los primeros cuatro problemas) o por otro lado, este trabaja con una mejor razón de éxito que el UMDAD (función F_{Cuban1}). En la primera columna de la tabla 4.3 se presenta el problema a ser resuelto. La segunda columna (encabezamiento dUMDAD) muestra para cada problema, el par (r,m) donde dUMDA supera a la versión centralizada, más el resultado de las corridas con todas las islas desconectadas (no existe migración entre las islas $(r=0,m=0)$). La tercera columna muestra los resultados para el par (r,m) elegido. La cuarta y quinta columnas presentan los parámetros y los resultados de las corridas de la versión centralizada, el UMDAD. En todos los casos, las columnas de los resultados muestran el porciento de convergencia y el promedio de las evaluaciones realizadas. Se recomienda a los lectores interesados revisar otros resultados [87].

Funciones	dUMDA con cuatro islas		UMDAD		p-value
$F_{IsoPeak}$, $n = 64$	$r = 1, m = 40$	98 %	$N = 3200$	97 %	
		45954.6 ± 2739.7		55158.7 ± 4424.7	0.0
	$r = 0, m = 0$	46 %	$N = 800$	10 %	
		54191.3 ± 3193.0		13360.0 ± 386.4	0.0
$F_{Plateau}$, $n = 600$	$r = 1, m = 30$	95 %	$N = 600$	100 %	
		17640 ± 627.4		22152.0 ± 550.5	0.0
	$r = 0, m = 0$	0 %	$N = 150$	0 %	
		-		-	-
$F_{Quadratic}$, $n = 66$	$r = 1, m = 12$	95 %	$N = 2000$	96 %	
		33452.6 ± 2448.4		34583.3 ± 2426.4	0.0015
	$r = 0, m = 0$	18 %	$N = 500$	0 %	
		34000.00 ± 0.00		-	-
F_{Muhl}, $n = 200$	$r = 1, m = 5$	95 %	$N = 1400$	96 %	
		39505.4 ± 1503.1		41183.3 ± 1646.3	0.0
	$r = 0, m = 0$	1 %	$N = 350$	0 %	
		43400.0 ± 0.0		-	-
F_{Cuban1}, $n = 21$	$r = 2, m = 10$	92 %	$N = 800$	46 %	
		5408.7 ± 521.9		5043.4 ± 372.1	0.00004
	$r = 0, m = 0$	53 %	$N = 200$	45 %	
		8256.6 ± 1535.4		1355.5 ± 84.1	0.0

Tabla 4.3: Porciento de éxito más el promedio del número de evaluaciones para converger al óptimo utilizando un algoritmo dUMDAD con cuatro islas para la resolución de los problemas discretos. Los resultados se muestran para diferentes combinaciones de la frecuencia de migración (r), el número de emigrantes (m) y el tamaño de población (N)

¿Por qué dUMDA reduce el esfuerzo evaluativo?

A continuación se explican las razones que permiten al dUMDA reducir drásticamente el número de evaluaciones de la función objetivo (para el caso de selección por truncamiento, dos islas, y $r = 1$). La motivación inicial reside en el presupuesto, de que en cada generación la migración incrementa la aptitud promedio de la población. Esto implica que la respuesta a la selección [99] del algoritmo en cada isla se incrementa, con respecto al caso de no migración. La ganancia en la respuesta a la selección conduce a una aceleración de la convergencia en los dEDA, pero el balance que se requiere entre exploración y explotación hace que esta aceleración tenga un límite. Este análisis es válido y extensible para cualquier otra función de aptitud.

Sea $R(t)$, la respuesta a la selección de la generación t antes de la migración, entonces:

$$R(t) = \overline{f}(t+1) - \overline{f}(t)$$

donde $\overline{f}(t)$ y $\overline{f}(t+1)$ son la media de las evaluaciones de la función de aptitud en la población t y $t+1$.

Sea $\overline{f}_b(t+1)$, $\overline{f}_w(t+1)$ y $\overline{f}_r(t+1)$, los respectivos promedios de los valores de la función de aptitud para los subconjuntos de los mejores M, los peores M y los $(N-2\cdot M)$ restantes individuos de una población con tamaño N. En este análisis M es igual a m, es decir, el número de individuos que se intercambia entre las islas en el proceso de migración. Entonces:

$$\overline{f}(t+1) = \frac{M\cdot\overline{f}_b(t+1) + (N-2\cdot M)\cdot\overline{f}_r(t+1) + M\cdot\overline{f}_w(t+1)}{N}$$

Se asume que los valores $\overline{f}_b(t+1)$ de cada isla son similares. Esta suposición es correcta y ampliamente utilizada en la literatura, producto de que las islas son homogéneas y los subconjuntos de los mejores M (en cada isla) son una aproximación de un conjunto de individuos igualmente adaptados. Ahora se plantea el promedio de la población (después que tiene lugar la migración de los mejores M individuos y se reemplazan los M peores en las islas vecinas):

$$\overline{f}_{mig}(t+1) = \frac{2\cdot M\cdot\overline{f}_b(t+1) + (N-2\cdot M)\cdot\overline{f}_r(t+1)}{N}$$

Por consiguiente, la migración produce un incremento de la respuesta a la selección igual a:

$$\Delta R_{mig}(t) = R_{mig}(t) - R(t) = \overline{f}_{mig}(t+1) - \overline{f}(t+1) = \frac{M\cdot\left(\overline{f}_b(t+1) - \overline{f}_w(t+1)\right)}{N}$$

Advierta que si M se incrementa (acople alto), entonces $\overline{f}_b(t+1)$ tiende a decrecer, mientras $\overline{f}_w(t+1)$ tiende a crecer. Por tanto, $\left(\overline{f}_b(t+1) - \overline{f}_w(t+1)\right)$ decrece cuando M crece. Esto provoca una aceleración de la convergencia del algoritmo. Esto se observa en los resultados de los experimentos para F_{OneMax} (tabla 4.2): cuando M inicialmente crece, los resultados mejoran, pero posteriormente el porciento de éxito decrece considerablemente. Es importante recordar que los niveles en la respuesta a la selección del algoritmo sin migración no son suficiente para lograr la convergencia al óptimo con alta probabilidad. Finalmente, se observa que la magnitud $\left(\overline{f}_b(t+1) - \overline{f}_w(t+1)\right)$ depende del problema a optimizar, y así para cada problema, diferentes pares de (m,r) minimizan el número total de evaluaciones de la función de aptitud realizados por el dUMDA.

Por otro lado, si la frecuencia de migración r decrece, entonces decrecerá también la respuesta a la selección. En este caso, se observa un desaceleramiento de la convergencia, decreciendo también drásticamente el porciento de éxito pero con una exploración mejorada y adecuada para muchos problemas complejos. De alguna forma, otros estudios [165, 24] llegan a las mismas conclusiones pero por vías diferentes.

dUMDAC reduce el esfuerzo evaluativo

Luego de revisar los resultados para dominio discreto, en este apartado se hace un estudio similar sobre dominio continuo. La aplicación de algoritmos EDA paralelos o distribuidos, a la solución de problemas continuos es muy rara, debido a que no hay muchos trabajos realizados en esta área de estudio.

Funciones	*dUMDAC* con cuatro islas		*UMDAC*		p-value
$F_{Sphere}, n = 100$	$r = 1, m = 45$	100 %	$N = 600$	100 %	
		55002.0 ± 330.8		75630.0 ± 421.0	0.0
	$r = 0, m = 0$	45 %	$N = 150$	16 %	
		78000.0 ± 3167.1		20820.0 ± 2097.1	-
$F_{Ackley}, n = 100$	$r = 1, m = 20$	97 %	$N = 400$	100 %	
		45430.9 ± 315.3		62832.0 ± 358.1	0.0
	$r = 0, m = 0$	0 %	$N = 100$	0 %	
		-		-	-
$F_{Griewangk}, n = 100$	$r = 1, m = 20$	99 %	$N = 400$	100 %	
		29301.0 ± 281.9		40336.0 ± 315.1	0.0
	$r = 0, m = 0$	0 %	$N = 100$	0 %	
		-		-	-
$F_{Water}, n = 100$	$r = 1, m = 45$	100 %	$N = 800$	99 %	
		64856.0 ± 498.9		128945.4 ± 2941.3	0.0
	$r = 0, m = 0$	0 %	$N = 200$	0 %	
		-		-	-

Tabla 4.4: Porciento de éxito más el promedio del número de evaluaciones para converger al óptimo utilizando un algoritmo dUMDAC con cuatro islas para la resolución de los problemas continuos. Los resultados se muestran para diferentes combinaciones de la frecuencia de migración (r), el número de emigrantes (m) y el tamaño de población (N)

La tabla 4.4 muestra los resultados de la ejecución del dUMDAC sobre cuatro funciones continuas. Note que en todos los casos se confirman los resultados precedentes para dominio discreto, esto es, el algoritmo dUMDAC mejora los resultados del UMDAC. En el conjunto de funciones continuas los resultados son más relevantes: se tiene una clara y mayor reducción del número de evaluaciones de la función objetivo que en el caso del dominio discreto.

Nótese que cuando el dUMDAC no realiza migración y el UMDAC se ejecuta con el mismo tamaño de población que el de una isla, el porciento de convergencia es muy pequeño o nulo. Esta tabla corrobora los resultados que se presentan para dominio discreto: el aislamiento completo no es una técnica interesante desde el punto de vista de la eficiencia evaluativa. Todos los p-values muestran un alto nivel de diferencia significativa entre los resultados.

En este sentido es necesario insistir en el grado de dificultad de la función F_{Water} para la cual la versión descentralizada reduce el número de evaluaciones drásticamente (mucho más intenso que para los otros problemas). El dUMDAC aparece como una alternativa eficiente a los métodos matemáticos tradicionales y algoritmos mejorados para solucionar este problema.

Análisis del tiempo total de ejecución

En este apartado, se muestra como el algoritmo dUMDA (discreto y continuo) puede mejorar los resultados en términos del tiempo de ejecución utilizado, para resolver el mismo problema que el UMDA. Para esto se utiliza la fórmula de ganancia en velocidad (1.5.3).

Para evaluar el speed-up se emplea la taxonomía propuesta por Alba [4]. La misma divide el análisis del speed-up en dos tipos, speed-up *fuerte - tipo I - y* speed-up *débil, - tipo II* -. El speed-up fuerte tiene la intención de comparar el algoritmo paralelo con el mejor algoritmo secuencial existente, para resolver el problema. La segunda medida compara el algoritmo paralelo con su propia versión secuencial, siendo más pragmático en otros trabajos [25, 24]. Antes de la ejecución del algoritmo, se define la condición de parada (encontrar el óptimo) y me mide el speed-up (*tipo II.A*). En este trabajo se selecciona la medida débil del speed-up (*tipo II.A.2*), llamada *ortodoxa*. Este tipo de análisis ejecuta el mismo algoritmo (dUMDA) con cuatro islas sobre uno, dos y cuatro procesadores para poder hacer una comparación justa. El algoritmo que se ejecuta sobre un procesador no es la versión centralizada del UMDA, sino que es el dUMDA en sí (en los estudios paralelos no está permitido, ni es justo, comparar diferentes algoritmos). Se muestra que el tiempo de ejecución decrece a medida que aumenta el número de procesos en la resolución del problema. Todas las pruebas fueron ejecutadas en cuatro Pentium 4, a 2.4 GHz y 512 MB de memoria, ejecutando el sistema operativo RedHat Linux 7.2 e interconectadas por una red Gigabit Ethernet.

La tabla 4.5 muestra los resultados del speed-up en los dominios discreto y continuo. Note que en el caso discreto, especialmente para los problemas que necesitan un elevado tamaño de población (funciones $F_{IsoPeak}$ y $F_{Quadratic}$), el speed-up es superlineal. La razón se encuentra en la naturaleza estocástica de los algoritmos, donde el asincronismo permite ejecutar la búsqueda de forma diferente, dependiendo del número de procesadores que se utilice.

Para los problemas continuos, todos los resultados son superlineales. En este caso se tiene la misma explicación que para los problemas discreto, con una reducción adicional por la utilización de números reales, codificados en expresiones más complejas y necesitan estructuras de datos en memoria de mayor tamaño (más rápidas para trabajar en paralelo cuando se divide el trabajo entre los procesos).

Función	1-procesador	2-procesadores	4-procesadores
Dominio discreto			
F_{OneMax}	1.00	1.65	2.28
$F_{IsoPeak}$	1.00	10.55	13.75
$F_{Plateau}$	1.00	2.71	3.13
$F_{Quadratic}$	1.00	6.40	7.51
F_{Muhl}	1.00	3.19	3.62
Dominio continuo			
F_{Sphere}	1.00	4.94	6.43
F_{Ackley}	1.00	5.86	7.96
$F_{Griewangk}$	1.00	5.57	7.47
F_{Water}	1.00	3.35	6.82

Tabla 4.5: Resultados de la ganancia en velocidad (speed-up) para dominio discreto y continuo

4.5.2. Resultados del modelo celular

En este apartado se reportan y analizan los resultados obtenidos en la experimentación con el modelo cEDA.

El cUMDA sincrónico supera numéricamente al UMDA centralizado

Primeramente se estudia el comportamiento numérico del cUMDA frente al UMDA en los problemas F_{OneMax} (expresión A.1), $F_{Plateau}$ (expresión A.2), $F_{IsoPeak}$ (expresión A.3) que se encuentran descritos en el anexo A. Se evalúan cuatro propuestas de algoritmos cUMDA, dos utilizando el vecindario C25 con uno y cuatro individuos por celda ($C25-1\times1-20\times20$ y $C25-2\times2-10\times10$) y dos utilizando el vecindario C41 con uno y cuatro individuos por celda ($C41-1\times1-20\times20$ y $C41-2\times2-10\times10$). En todos los casos el tamaño de la población global es de 400 individuos (igual que para el UMDA centralizado). Una nota importante es que todas las propuestas utilizan mutación.

Para obtener resultados estadísticamente significantes se ejecutaron 100 corridas para cada prueba realizada. Posteriormente se computa el test de varianza ANOVA o Kruskal-Wallis para comparar los resultados. Para los tests se considera un nivel de significación del 95 % (p-value menor que 0.05). Tanto la versión centralizada como la celular están desarrolladas en C++ y se ejecutan en un Pentium 4, a 2.4 HGz, con sistema operativo RedHat Linux 7.2 y 512 MB de memoria. La tabla 4.6 muestra los parámetros utilizados en las corridas de los algoritmos estudiados.

Parámetro	Valor
Tamaño de población	400 individuos
Selección	Selección por truncamiento, $\tau = 0{,}5$
Mutación	Inversión de bit, $p_m = 1/n$
Reemplazamiento	Reemplazar si es mejor (elitista)
Condición de parada	Encontrar el óptimo o realizar 400000 evaluaciones
Algoritmo miembro	UMDA
Alternativa celular	Vecindario: $C25$ y $C41$
	Forma de población: $1\times1-20\times20$ y $2\times2-10\times10$

Tabla 4.6: Parámetros utilizados en los experimentos

cUMDA tiene mayor eficacia que UMDA La eficacia se define en base al número de veces que el algoritmo alcanza el óptimo en k corridas (para estos experimentos $k = 100$). El UMDA no logra encontrar el óptimo en ninguna corrida para el problema $F_{IsoPeak}$. Por el contrario, las cuatro versiones celulares no encuentran dificultades para alcanzar el óptimo en todas las corridas (un 100 % de éxito).

cUMDA tiene mayor eficiencia que UMDA La eficiencia se define en función del número de evaluaciones que realiza el algoritmo evolutivo, mientras menos evaluaciones realice, mayor eficiencia tiene el algoritmo. En la tabla 4.7 se presenta el promedio de evaluaciones realizadas y la desviación estándar que se necesita por cada uno de los cinco algoritmos estudiados en la solución de cada problema. En la última fila se presentan los resultados del análisis estadístico que comparan cada

Vecindario	F_{OneMax}	$F_{Plateau}$	$F_{IsoPeak}$
UMDA	26072.00± 463.20	18972.00±1054.86	-
cUMDA $25-1\times1-20\times20$	23298.11±382.76	16337.83±584.86	176878.71±36384.01
cUMDA $25-2\times2-10\times10$	22037.60±346.74	14961.16±511.65	218190.00±47518.06
cUMDA $41-1\times1-20\times20$	22500.89±361.14	15454.92±510.05	**176138.77**±41834.80
cUMDA $41-2\times2-10\times10$	**21851.72**±340.30	**14773.64**±469.24	253725.44±58172.59
p − value	+	+	+

Tabla 4.7: Eficiencia evaluativa de los algoritmos estudiados

uno de los cinco algoritmos. El signo de "+" implica que existe diferencia significativa, entre al menos dos de los algoritmos, para el problema analizado. Como se observa, el algoritmo más eficiente para cada problema (valores en negrita) es siempre una de las versiones celulares estudiadas.

Extensión del concepto de *tiempo de apoderamiento* al entorno de los cEDA Finalmente se incluye en esta sección un breve estudio del comportamiento teórico de los algoritmos propuestos en términos de su presión selectiva. En el caso de los cEA la presión selectiva se relaciona con el concepto de tiempo de apoderamiento y se define como el tiempo que toma a un individuo (el mejor) colonizar la población completa, con copias de él mismo, utilizando solo el operador de selección. Mientras más corto es el tiempo de apoderamiento, mayor presión selectiva y por ende, menos diversidad.

Para el cálculo del tiempo de apoderamiento en cEDA, inicialmente se inserta de forma aleatoria, un buen individuo ($aptitud = 1$) en una de las celdas de la población y el resto de las celdas se rellenan con malos individuos ($aptitud = 0$). Posteriormente se aplica el operador de selección, en este caso el truncamiento y se computa la proporción que existe de ese mejor individuo en el conjunto seleccionado. Esta proporción, en forma de distribución de probabilidad, es utilizada para muestrear los nuevos individuos que reemplazarán a los viejos, utilizando un mecanismo elitista. Esencialmente, este tipo de reemplazo se utiliza para no perder a ese buen individuo en las generaciones iniciales, donde su proporción es muy pequeña.

En la figura 4.4 se grafican las curvas de crecimiento para el UMDA y las cuatro alternativas de cEDA. Para obtener resultados válidos se extiende el tamaño de población a 64000 individuos. Como se observa, la mayor presión selectiva (tiempos de apoderamiento menores) corresponde al UMDA. Con respecto a las versiones celulares, los dos algoritmos que tienen un individuo por celda tienen menor presión selectiva (mayores tiempos de apoderamiento). La razón es, que en el caso de los cUMDA que tienen cuatro individuos por celda, las posibilidades de exploración del algoritmo se penalizan debido a que el número de individuos en el vecindario es multiplicado por cuatro.

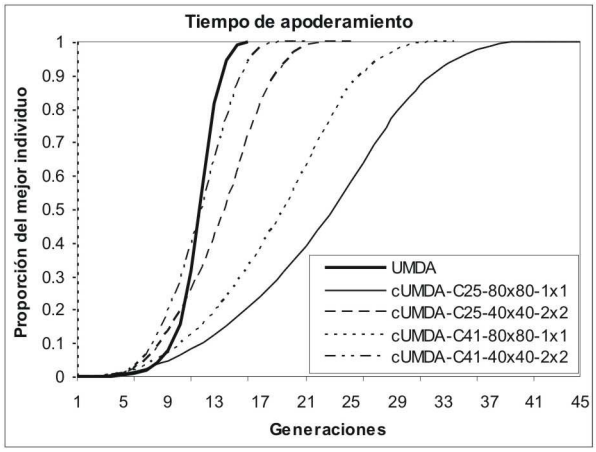

Figura 4.4: Curvas del tiempo de apoderamiento para los algoritmos estudiados

4.6. Conclusiones

En este capítulo se realizó un estudio de los modelos descentralizados en EDA, con aplicación específica al algoritmo UMDA, aunque los resultados pueden ser generalizados a otros EDA.

La primera propuesta consiste en un modelo de islas distribuidas de forma asincrónica sobre un grupo de procesadores, realizando migraciones cada cierto número de generaciones. El segundo modelo propuesto fue el celular, donde las poblaciones quedan estructuradas en forma de rejilla y donde la selección se realiza entre sub-poblaciones vecinas. Ambos tienen la característica de ejecutar un algoritmo EDA en cada población descentralizada. Como principales resultados del capítulo se tienen los siguientes:

1. Análisis de los modelos descentralizados y su aplicación al entorno de los EDA, dando lugar a dos nuevas clases de algoritmos llamadas dEDA (EDA distribuido) y cEDA (EDA celular).
2. Propuesta del algoritmo dUMDA como implementación del modelo distribuido en EDA para dominio discreto y continuo.
3. Propuesta del algoritmo cUMDA como implementación del modelo celular en EDA discretos.
4. Utilización de la respuesta a la selección, para explicar de forma teórica por qué el dUMDA reduce el esfuerzo evaluativo.

Desde el punto de vista experimental se obtuvieron los siguientes resultados:

1. Primero se muestra que el dUMDA reduce el esfuerzo evaluativo realizado por el UMDA, logrando reducir de forma drástica el número de evaluaciones realizadas para diferentes problemas, tanto discretos como continuos.
2. Desde el punto de vista del tiempo de ejecución, se muestra que el dUMDA discreto alcanza ganancias de velocidades sub-lineales, excepto en los problemas $F_{IsoPeak}$ y $F_{Quadratic}$, donde la ganancia en velocidad es superlineal (supera al número de procesadores). En el caso continuo, el dUMDAC reduce drásticamente el tiempo de ejecución, demostrándose mediante las ganancias superlineales en todos los problemas estudiados.

3. En el caso del modelo celular se muestra que el cUMDA supera numéricamente al UMDA para los problemas estudiados en cuanto a eficacia y eficiencia.
4. Se extiende el concepto de tiempo de apoderamiento al entorno de los cEDA, como argumento teórico para el estudio de la presión selectiva.

CONCLUSIONES

Cada uno de los capítulos anteriores contiene una discusión detallada de las conclusiones correspondientes (secciones 2.10, 3.8 y 4.6) por lo que en este apartado se plantean los resultados y conclusiones de la tesis desde una perspectiva más general. Además, se enuncian las recomendaciones y el trabajo futuro de la investigación.

La investigación desarrollada en esta tesis se ha centrado en la eficiencia de los EDA, tanto evaluativa como en tiempo de ejecución. Desde el punto de vista del paralelismo, se ilustra una vez más, que su utilización es una alternativa a tener en cuenta a la hora de abordar problemas costosos computacionalmente. Más que eso, se muestra la utilidad de las arquitecturas distribuidas como máquinas paralelas eficientes y de bajo costo económico.

Los resultados más importantes son los siguientes:

1. Se crea la clase de algoritmos basados en restricciones, CBEDA, la cual está formada por $\text{CBEDA}_{\text{MMHC}}$, $\text{CBEDA}_{\text{TPDA}}$ y $\text{CBEDA}_{\text{CMMHC}}$ (capítulo 2). Se caracteriza el comportamiento numérico de los CBEDA creados, utilizando el método de las B-funciones.
2. Se crean dos versiones paralelas del algoritmo PADA: una con balance de los cálculos de las dependencias marginales y condicionales, $\text{pCBEDA}_{\text{LPA}-\text{BAL}}$ y otra que no utiliza el balance, $\text{pCBEDA}_{\text{LPA}-\text{UNB}}$ (capítulo 3).
3. Se crean dos algoritmos EDA paralelos basados en restricciones, $\text{pCBEDA}_{\text{MMHC}}$ y $\text{pCBEDA}_{\text{TPDA}}$ (capítulo 3).
4. Se crean dos algoritmos EDA descentralizados, dEDA y cEDA, basados en el modelo de islas y el celular, respectivamente (capítulo 4). Se explica, teóricamente, por qué el modelo de islas basado en UMDA y con selección por truncamiento, reduce el esfuerzo evaluativo. El algoritmo de islas para el UMDA es eficiente en tiempo, con ganancias de velocidad superlineales.

Las conclusiones más importantes son las siguientes:

1. Se muestra que los métodos de aprendizaje de redes Bayesianas basados en pruebas de independencia son efectivos para crear algoritmos EDA eficientes, tanto en el dominio discreto como en el continuo. Los algoritmos $\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{TPDA}}$ son más eficientes en número de evaluaciones que las variantes secuenciales del estado del arte, basadas en optimización de métricas en el conjunto de funciones utilizadas (capítulo 2).
2. La comparación entre los algoritmos $\text{CBEDA}_{\text{MMHC}}$ y $\text{CBEDA}_{\text{TPDA}}$, muestra que ambos tienen su espacio, o sea, en algunos problemas es mejor el $\text{CBEDA}_{\text{MMHC}}$ y viceversa, lo cual depende de las características del problema de optimización (capítulo 2).

3. Se comprueba que las variantes secuenciales de los algoritmos estudiados, contienen suficientes elementos que se calculan independientemente. Por lo que las variantes paralelas creadas en esta tesis conllevan a una reducción del tiempo de ejecución (capítulo 3).
4. La versión paralela balanceada de PADA tiene mayor ganancia en velocidad y mayor eficiencia que la no balanceada. Parece que al terminar los cálculos de las dependencias marginales el número de aristas resultantes es sustancialmente diferente en cada procesador. Al balancear estos cálculos se aprovecha mejor el tiempo de cómputo de los procesos (capítulo 3).
5. Los modelos descentralizados aumentan la eficiencia evaluativa del EDA por la forma de estructurar la población y la interacción entre los diferentes algoritmos. El modelo de islas con UMDA no solo conlleva a una mayor eficiencia evaluativa sino que también reduce el tiempo de ejecución (capítulo 4).

RECOMENDACIONES

Las recomendaciones fundamentales a la comunidad científica, que trabaja en el área de los algoritmos EDA, es que debe tener en cuenta las potencialidades de los métodos basados en pruebas de independencia. Además, que al utilizar un algoritmo EDA para la resolución de un problema de optimización debe comenzar por los modelos probabilísticos más simples, como el de independencia o poliárbol (cuyo aprendizaje es menos costoso), e ir escalando a modelos más generales en correspondencia con la dificultad del problema.

Se pueden identificar como líneas de trabajo futuro las siguientes:

1. Investigar el comportamiento del modelo de islas y el celular con la clase de algoritmos CBEDA.
2. Estudiar cómo escalan los algoritmos propuestos al aumentar el número de variables.
3. Paralelizar algoritmos de aprendizaje que construyen mezclas de distribuciones, comenzando por las mezclas de árboles e ir escalando a modelos más generales.

REFERENCIAS BIBLIOGRÁFICAS

[1] S. Acid y L. M. de Campos. Approximation of Causal Networks by Polytrees: an Empirical Study. En B. Bouchon-Meunier, R. R. yager, y L. A. Zadeh, editores, *Advances in Intelligent Computing*, Lecture Notes in Computer Science 945, páginas 149–158, 1995. Springer-Verlag, Berlin, 1995.

[2] C. W. Ahn, D. E. Goldberg, y R. Ramakrishna. Multiple-deme parallel estimationof distribution algorithms: Basic framework and application. Informe Técnico 2003016, University of Illinois at Urbana-Champaign, 2003.

[3] H. Akaike. A new look at the statistical identification model. *IEEE Transactions on Automatic Control*, páginas 716–723, 1974.

[4] E. Alba. Parallel Evolutionary Algorithms can Achieve Superlinear Performance. *Information Processing Letters*, tomo 82(1):7–13, 2002.

[5] E. Alba, J. Madera, B. Dorronsoro, A. Ochoa, y M. Soto. Theory and Practice of Cellular UMDA for Discrete Optimization. En T. P. Runarsson, H.-G. Beyer, E. K. Burke, J. J. M. Guervós, L. D. Whitley, y X. Yao, editores, *PPSN*, tomo 4193 de *Lecture Notes in Computer Science*, páginas 242–251. Springer, 2006. ISBN 3-540-38990-3.

[6] E. Alba y M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, tomo 6(5):443–462, 2002.

[7] E. Alba y J. M. Troya. A Survey of Parallel Distributed Genetic Algorithms. *Complexity*, tomo 4(4):31–52, 1999. John Wiley & Sons, Inc.

[8] E. Alba y J. M. Troya. Influence of the Migration Policy in Parallel Distributed GAs with Structured and Panmictic Populations. *Applied Intelligence*, tomo 12(3):163–181, 2000.

[9] E. Alba y J. M. Troya. Analyzing Synchronous and Asynchronous Parallel Distributed Genetic Algorithms. *Future Generation Computer Systems*, tomo 17(4):451–465, 2001.

[10] R. Arderí. *Algoritmo con Estimación de Distribuciones con Cópula Gaussiana*. Proyecto Fin de Carrera, Universidad de la Habana, 2007.

[11] S. Baluja. Structure and performance of fine-grain parallelism in genetic search. En S. Forrest, editor, *6th ICGA*, páginas 155–162. Morgan Kaufmann, 1993.

[12] S. Baluja. Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Informe Técnico CMU-CS-94-163, Carnegie Mellon University, 1994.

[13] S. Baluja. An empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics. Informe Técnico CMU-CS-95-193, Carnegie Mellon University, 1995.

[14] S. Baluja y S. Davies. Fast Probabilistic Modeling for Combinatorial Optimization. En *AAAI-98*, 1998.

[15] E. Bengoetxea. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. Tesis Doctoral, Ecole Nationale Supérieure des Télécommunications, 2002. Paris, France.

[16] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, y C. Boeres. Inexact Graph Matching Using Learning and Simulation of Bayesian Networks. An Empirical Comparison between Different Approaches with Synthetic Data. En *Workshop Notes of CaNew2000: Workshop on Bayesian and Causal Networks: From Inference to Data Mining*, 2000. Fourteenth European Conference on Artificial Intelligence, ECAI2000. Berlin.

[17] L. Bilmes. Dynamic Bayesian Multinets. En *Proceedings of the 16th conference on Uncertainty in Artificial Intelligence*, páginas 38–45. Morgan Kaufmann Publishers, 2000.

[18] I. O. Bohachevsky, M. E. Johnson, y M. L. Stein. Generalized Simulated Annealing for Function Optimization. *Technometrics*, tomo 28(3):209–217, 1986.

[19] C. Branden y J. Tooze. *Introduction to Protein Structure*. Garland, New York, 1998.

[20] L. Brown, I. Tsamardinos, y C. Aliferis. A novel algorithm for scalable and accurate bayesian network learning. En *11th World Congress on Medical Informatics (MEDINFO)*, 2004. California.

[21] L. Brown, I. Tsamardinos, y C. F. Aliferis. A Comparison of Novel and State-of-the-Art Polynomial Bayesian Network Learning Algorithms. En M. M. Veloso y S. Kambhampati, editores, *AAAI*, páginas 739–745. AAAI Press AAAI Press / The MIT Press, 2005. ISBN 1-57735-236-X.

[22] W. Buntine. A guide to the literature on learning probabilistic networks from data. *Knowledge and Data Engineering*, tomo 8:195–210, 1996.

[23] D. R. Butenhof. *Programing with POSIX Threads*. Addison-Wesley Professional Computing Series, 1997.

[24] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Press, 2000.

[25] E. Cantú-Paz y D. E. Goldberg. Predicting Speedups of Idealized Bounding Cases of Parallel Genetic Algorithms. En T. Bäck, editor, *Proceedings of the Seventh International Conference on GAs*. Morgan Kaufmann, 1997. San Francisco.

[26] J. Cheng, R. Greiner, J. Kelly, D. A. Bell, y W. Liu. Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence*, tomo 137:43–90, 2002.

[27] D. Chickering, D. Geiger, y D. Heckerman. Learning Bayesian networks: search methods and experimental results. En *Proceeding of 5^{th} Conference Artificial Intelligence and Statistics*, páginas 112–128, 1995.

[28] D. Chickering, C. Meek, y D. Heckerman. Large-Sample Learning of Bayesian Networks is NP-Hard. *Journal of Machine Learning Research*, tomo 5:1287–1330, 2004.

[29] C. K. Chow y C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, tomo 14:462–467, 1968.

[30] J. P. Cohoon, S. U. Hedge, W. Martin, y D. Richards. Parallel Genetic Algorithms for Hypercube. En J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, páginas 148–154, 1987. Hillsdale, NJ: Lawrence Erlbaum Associates.

[31] C. Cotta. Protein structure prediction using evolutionary algorithms hybridized with backtracking. En J. Mira y J. R. Alvarez, editores, *Artificial Neural Nets Problem Solving Methods*, tomo 2687 de *Lecture Notes in Computer Science*, páginas 321–328. Berlin, Germany: Springer Verlag, 2003.

[32] C. Cotta, E. Alba, R. Sagarna, y P. Larrañaga. Adjusting weights in artificial neural networks using evolutionary algorithms. En P. Larrañaga y J. A. Lozano, editores, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[33] V. Cutello, G. Nicosia, M. Pavone, y J. Timmis. An immune algorithm for protein structure prediction on lattice models. *IEEE Trans. Evol. Comput.*, tomo 11(1):101–117, 2007.

[34] J. S. De Bonet, C. L. Isbell, y P. Viola. MIMIC: Finding Optima by Estimating Probability Densities. *Advances in Neural Information Processing Systems*, tomo 9, 1997.

[35] L. M. de Campos. Independency relationship and learning algorithms for singly connected networks. *Experimental and Theoretical Artificial Intelligence*, tomo 10:511–549, 1998.

[36] L. M. de Campos y J. Huete. Algorithms for Learning Decomposable Models and Chordal Graphs. En *Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, páginas 46–53. Morgan Kaufmann, San Francisco, CA, 1997.

[37] K. Deb y D. E. Goldberg. Analyzing deception in trap functions. En *Analyzing deception in trap functions. Foundations of Genetic Algorithms 2*, páginas 93–108, 1993.

[38] K. A. Dill. Dominant forces in protein folding. *Biochemistry*, tomo 24(1501), 1985.

[39] R. Etxeberria y P. Larrañaga. Global optimization with Bayesian networks. En *II Symposium on Artificial Intelligence. CIMAF99. Special Session on Distributions and Evolutionary Optimization*, páginas 332–339, 1999.

[40] S. E. Fienberg. *The Analysis of cross–classified categorical data*. MIT Press, 1981.

[41] L. J. Fogel. Autonomous automata. *Industrial Research*, tomo 4:14–19, 1962.

[42] M. Forum. MPI: A Message-Passing Iterface Standard. *The International Journal of Supercomputer Applications and High Performance Computing*, tomo 8 (3/4):119–416, 1994.

[43] N. Friedman y M. Goldszmidt. Learning Bayesian networks with local structure. En *Proceedings of the twelfth international conference on uncertainty in artificial intelligence*, 1996.

[44] E. Galić y M. Höhfeld. Improving the generalization performance of multi-layer-perceptrons with population-based incremental learning. En *Parallel Problem Solving from Nature. PPSN-IV*, páginas 740–750, 1996.

[45] M. R. Gallagher. Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling. Informe Técnico Doctoral Thesis, Department of Computer Science and Electrical Engineering, University of Queensland, 2000.

[46] J. A. Gámez, J. L. Mateo, y J. M. Puerta. EDNA: Estimation of Dependency Networks Algorithm. En *IWINAC '07: Proceedings of the 2nd international work-conference on The Interplay Between Natural and Artificial Computation, Part I*, páginas 427–436. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-73052-1.

[47] J. A. Gámez, J. L. Mateo, y J. M. Puerta. Improved EDNA (estimation of dependency networks algorithm) using combining function with bivariate probability distributions. En *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, páginas 407–414. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-130-9.

[48] M. Giacobini, E. Alba, y M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. En *GECCO*, páginas 955–966. Springer Verlag, 2003.

[49] C. Glymour y G. F. Cooper, editores. *Computation, Causation, and Discovery*. AAAI Press/The MIT Press, 1999.

[50] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading MA, 1989.

[51] D. E. Goldberg. *The Design of Innovation*. Kluwer Academic Publishers, 2002.

[52] D. E. Goldberg. The Design of Innovating Machines: A Fundamental Discipline for a Postmodern Systems Engineering. En *Engineering Systems Symposium*. MIT Engineering Systems Division, USA, 2004.

[53] J. Grahl y F. Rothlauf. PolyEDA: Combining Estimation of Distribution Algorithms and linear inequality constraints. En *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, páginas 1174–1185, 2004.

[54] J. J. Grefenstette. Parallel Adaptative Algorithms for Function Optimization. Informe Técnico CS-81-19, Vanderbilt University, 1981.

[55] T. Groppo. *Learning Bayesian Networks Skeleton: A Comparison Between TPDA and PMMS Algorithm*. Proyecto Fin de Carrera, Universite Claude Bernard Lyon I, 2006.

[56] H. Handa. Hybridization of Estimation of Distribution Algorithms with a Repair Method for Solving Constraint Satisfaction Problems. En E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, y J. F. Miller, editores, *GECCO*, tomo 2723 de *Lecture Notes in Computer Science*, páginas 991–1002. Springer, 2003. ISBN 3-540-40602-6.

[57] H. Handa. Estimation of Distribution Algorithms with mutation. En G. R. Raidl y J. Gottlieb, editores, *EvoCOP*, tomo 3448 de *Lecture Notes in Computer Science*, páginas 112–121. Springer, 2005.

[58] G. Harik, F. G. Lobo, y D. E. Golberg. The Compact Genetic Algorithm. En *Proceedings of the IEEE Conference on Evolutionary Computation*, páginas 523–528, 1998.

[59] D. Heckerman. A tutorial on learning with Bayesian networks. Informe Técnico MSR-TR-95-06, Redmond, WA, USA, 1995.

[60] D. Heckerman. Bayesian networks for knowledge discovery. En *Advances in Knowledge Discovery and Data Mining*, páginas 273–305, 1996.

[61] D. Heckerman, D. Geiger, y D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, tomo 20:197–243, 1995.

[62] D. Heckerman, C. Meek, y G. Cooper. A Bayesian Approach to Causal Discovery. En C. Glymour y G. Cooper, editores, *Computation, Causation, and Discovery*. AAAI Press, Menlo Park, California, 1999.

[63] M. Henrion. Propagating Uncertainty in Bayesian Networks by Probabilistic Logic Sampling. *Uncertainty in Artificial Intelligence*, tomo 2:317–324, 1988.

[64] S. Höfinger, T. Schindler, y A. Aszod. Parallel Global Optimization of High-Dimensional Problems. En *Lecture Notes in Computer Science*, tomo 2474, 2002.

[65] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1992.

[66] M. Jordan. *Learning in Graphical Models*. MIT Press, 1999.

[67] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[68] V. Kumar, A. Grama, A. Gupta, y G. Karypis. *Introduction to parallel computing. Design and analysis of algorithms*. The Benjamin/Cummings Publishing Company, Inc., 1994.

[69] L. D. la Ossa, J. A. Gámez, y J. M. Puerta. Migration of probability models instead of individuals: An alternative when applying the island model to EDAs. tomo 3242 de *Lecture Notes in Computer Science*. Springer, 2004.

[70] W. Lam y F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, tomo 10(4), 1994.

[71] P. Larraaga y J. A. Lozano, editores. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[72] P. Larrañaga. A Review on Estimation of Distribution Algorithms. En P. Larrañaga y J. A. Lozano, editores, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[73] P. Larrañaga, R. Etxeberria, J. A. Lozano, y J. M. Peña. Optimization by learning and simulation of Bayesian and Gaussian networks. Informe Técnico KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.

[74] P. Larrañaga, R. Etxeberria, J. A. Lozano, y J. M. Peña. Combinatorial optimization by learning and simulation of Bayesian networks. En *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, páginas 343–352, 2000. Stanford.

[75] P. Larrañaga, R. Etxeberria, J. A. Lozano, y J. M. Peña. Optimization in continuous domains by learning and simulation of Gaussian networks. En A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, páginas 201–204, 2000.

[76] P. Larrañaga, R. Etxeberria, J. A. Lozano, B. Sierra, I. Inza, y J. M. Peña. A review of the cooperation between evolutionary computation and probabilistic graphical models. En *Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, páginas 314–324, 1999. La Habana.

[77] S. L. Lauritzen. *Graphical Models*. Oxford:Clarendon Press, 1996.

[78] C. F. Lima, M. Pelikan, D. E. Goldberg, F. G. Lobo, K. Sastry, y M. Hauschild. Influence of Selection and Replacement Strategies on Linkage Learning in BOA. En *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, páginas 1083–1090. IEEE Press, 2007.

[79] F. G. Lobo, C. F. Lima, y H. Mártires. An Architecture for Massively Parallelization of the Compact Genetic Algorithm. En *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, 2004.

[80] J. A. Lozano, P. Larrañnaga, I. Inza, y E. Bengoetxea, editores. *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, tomo 192 de *Studies in Fuzzines and Soft Computing*. Springer, 2006.

[81] J. A. Lozano, R. Sagarna, y P. Larrañaga. Parallel Estimation of Bayesian Networks Algorithms. En *Third Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, 2001. La Habana.

[82] J. A. Lozano, R. Sagarna, y P. Larrañaga. Parallel Estimation of Distribution Algorithms. En P. Larrañaga y J. A. Lozano, editores, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academis Publishers, 2002.

[83] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. ISBN 1558603484.

[84] J. Madera, E. Alba, y G. Luque. Performance Evaluation of the Parallel Polytree Approximation Distribution Algorithm on Three Network Technologies. En *Procs. of CACIC*, 2006. Argentina.

[85] J. Madera, E. Alba, y G. Luque. Performance Evaluation of the Parallel Polytree Approximation Distribution Algorithm on Three Network Technologies. *Revista Iberoamericana de Inteligencia Artificial*, tomo 11(35), 2007.

[86] J. Madera, E. Alba, y A. Ochoa. Parallel Estimation of Distribution Algorithms. En E. Alba, editor, *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons, Inc., 2005.

[87] J. Madera, E. Alba, y A. Ochoa. A Parallel Island Model for Estimation of Distribution Algorithms. En I. I. J. A. Lozano, P. Larrañaga y E. Bengoetxea, editores, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*. Springer, 2006. To Appear.

[88] J. Madera y B. Dorronsoro. Estimation of Distribution Algorithms. En E. Alba y R. Martí, editores, *Metaheuristics Procedures for Training Neural Networks*. Springer, 2006. To Appear.

[89] J. Madera y A. Ochoa. Un EDA basado en aprendizaje MAX-MIN con escalador de colinas. Informe Técnico 2006-396, Instituto de Cibernética, Matemática y Física (ICIMAF), 2006.

[90] T. Mahnig y H. Mühlenbein. Comparing the adaptive Boltzmann selection schedule SDS to truncation selection. En *Third Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, páginas 121–128, 2001. La Habana.

[91] T. Mahnig y H. Mühlenbein. Optimal mutation rate using Bayesian priors for Estimation of Distribution Algorithms. En *Lectures Notes in Computer Science*, tomo 2264, páginas 33–48. Springer Verlag, 2001.

[92] B. Manderick y P. Spiessens. Fine-grained parallel genetic algorithm. En *3rd ICGA*, páginas 428–433, 1989.

[93] B. Maxwell y S. Anderson. Training Hidden Markov Models using Population-Based Learning. En *Genetic and Evolutionary Computation Conference, GECCO-99*, 1999.

[94] C. Meek. Strong Completeness and Faithfulnes in Bayesian Networks. En *Conference on Uncertainty in Artificial Intelligence*, 1995.

[95] A. Mendiburu. *Parallel Implementation of Estimation of Distribution Algorithms Based on Probabilistic Graphical Models. Application to Chemical Calibration Models*. Tesis Doctoral, University of the Basque Country, 2006.

[96] A. Mendiburu, J. Lozano, y J. Miguel-Alonso. Parallel Estimation of Distribution Algorithms: New Approaches. Informe Técnico EHU-KAT-IK-1-3, Department of Computer Architecture and Technology, The University of the Basque Country, 2003. Submitted to IEEE Transactions on Evolutionary Computation.

[97] A. Mendiburu, J. A. Lozano, y J. Miguel-Alonso. Parallel Implementation of EDAs based on Probabilistic Graphical Models. *IEEE Transactions on Evolutionary Computation*, tomo 9(4):406–423, 2005.

[98] A. Mendiburu, J. Miguel-Alonso, y J. A. Lozano. Implementation and performance evaluation of a parallelization of estimation of bayesian networks algorithms. Informe Técnico EHU-KAT-IK-XX-04, Department of Computer Architecture and Technology, The University of the Basque Country, 2004. Submitted to Parallel Computing.

[99] H. Mühlenbein. The Equation for Response to Selection and its Use for Prediction. *Evolutionary Computation*, tomo 5:303–346, 1998.

[100] H. Mühlenbein y T. Mahnig. Convergence Theory and Applications of the Factorized Distribution Algorithm. *Journal of Computing and Information Technology*, tomo 7:19–32, 1999.

[101] H. Mühlenbein y T. Mahnig. Evolutionary Algorithms: From Recombination to Search Distributions. *Theoretical Aspects of Evolutionary Computing. Natural Computing*, páginas 137–176, 2000.

[102] H. Mühlenbein y T. Mahnig. Evolutionary synthesis of Bayesian networks for optimization. En *Advances in Evolutionary Synthesis of Neural Systems*, páginas 429–455. MIT. Press, 2001.

[103] H. Mühlenbein, T. Mahnig, y A. Ochoa. Schemata, Distributions and Graphical Models in Evolutionary Optimization. *Journal of Heuristics*, tomo 5:215–247, 1999.

[104] H. Mühlenbein y G. Paaß. From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. En *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature - PPSN IV*, páginas 178–187, 1996.

[105] H. Mühlenbein y D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (BGA). *Evolutionary Computation*, tomo 1:335–360, 1993.

[106] H. Mühlenbein, M. Schomish, y J. Born. The parallel genetic algorithm as a function optimizer. *Parallel Computing*, tomo 17:619–632, 1991.

[107] H. Mühlenbein y H.-M. Voigt. Gene pool recombination in genetic algorithms. *Metaheuristics: Theory and applications*, páginas 53–62, 1996.

[108] R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[109] D. Nilsson. An Efficient Algorithm for Finding the M most Probable Configuration in Bayesian Networks. *Statistics and Computing*, tomo 2:159–173, 1998.

[110] J. Ocenasek. *Parallel Estimation of Distribution Algorithms*. Tesis Doctoral, Brno University of Technology, 2002.

[111] J. Ocenasek y J. Schwarz. The Parallel Bayesian Optimization Algorithm. En *In Proceedings of the European Symposium on Computational Inteligence*, páginas 61–67, 2000. Slovak Republic.

[112] J. Ocenasek y J. Schwarz. The Distributed Bayesian Optimization Algorithm for combinatorial optimization. En *In EUROGEN 2001 - Evolutionary Methods for Design, Optimisation and Control*, páginas 115–120, 2001. Athens, Greece.

[113] A. Ochoa. How to Deal with Costly Fitness Functions in Evolutionary Computation. En *Proceedings of the 13th ISPE/IEE International Conference on CAD/CAM. Robotics & Factories of the Future*, páginas 788–793, 1997.

[114] A. Ochoa, H. Mühlenbein, y M. Soto. Factorized Distribution Algorithm Using Bayesian Networks. En A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, páginas 212–215, 2000.

[115] A. Ochoa, H. Mühlenbein, y M. Soto. A Factorized Distribution Algorithm Using Single Connected Bayesian Networks. En M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, y H.-P. Schwefel, editores, *Parallel Problem Solving from Nature – PPSN VI. Lecture Notes in Computer Science 1917*, páginas 787–796, 2000.

[116] A. Ochoa y M. Soto. Partial Evaluation of Genetic Algorithms. En *Proceedings of the X International Conference on Industrial and Engineering Applications of AI and Expert Systems*, páginas 217–222, 1997. Atlanta, USA.

[117] A. Ochoa y M. Soto. B-functions: The class of random Boltzmann functions, 2006.

[118] A. Ochoa y M. Soto. Linking Entropy to Estimation of Distribution Algorithms. En J. A. Lozano, P. Larrañaga, I. Inza, y E. Bengoetxea, editores, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*. Springer, 2006. To Appear.

[119] A. Ochoa y M. Soto. On the preformance of the Bayesian Optimization Algorithm with B-functions. Informe Técnico 2006-383, Instituto de Cibernética, Matemática y Física (ICIMAF), 2006.

[120] A. Ochoa y M. Soto. On the preformance of the Bayesian Optimization Algorithm with B-functions. Informe Técnico 2006-383, ICIMAF, 2006.

[121] A. Ochoa, M. Soto, R. Santana, J. Madera, y N. Jorge. The Factorized Distribution Algorithm and the Junction Tree: A Learning Perspective. En *Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, páginas 368–377, 1999. La Habana.

[122] O. Ochoa y A. Ochoa. ViPoC: un clúster virtual, interactivo y portátil. Informe Técnico 2007-415, Instituto de Cibernética, Matemática y Física (ICIMAF), 2007.

[123] J. Pearl. *Probabilistic Reasoning in Inteligent Systems: Netowrks of Pausible Inference*. Morgan and Kaufmanm, 1988. ATT Bell Laboratories.

[124] J. Pearl. *Causality, Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[125] M. Pelikan. *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*, tomo 170 de *Studies in Fuzzines and Soft Computing*. Springer-Verlag, 2005.

[126] M. Pelikan y D. E. Goldberg. Hierarchical Problem Solving and the Bayesian Optimization Algorithm. En D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, y H.-G. Beyer, editores, *Proceedings of the Genetic and Evolutionary Computation Conference*, páginas 267–274. Morgan Kaufmann, 2000.

[127] M. Pelikan y D. E. Goldberg. Research on the Bayesian optimization algorithm. En A. S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, páginas 212–215, 2000.

[128] M. Pelikan, D. E. Goldberg, y E. Cantú-Paz. BOA: The Bayesian optimization algorithm. En W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, y R. E. Smith, editores, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, tomo 1, páginas 525–532. Morgan Kaufmann Publishers, San Francisco, CA, 1999. Orlando, FL.

[129] M. Pelikan, D. E. Goldberg, y F. G. Lobo. A Survey of Optimization by Building and Using Probabilistic Models. Informe Técnico IlliGAL Report 99018, University of Illinois at Urbana-Champaign, 1999.

[130] M. Pelikan y J. D. Laury. Order or Not: Does Parallelization of Model Building in hBOA Affect Its Scalability? Informe Técnico 2006005, Missouri Estimation of Distribution Algorithms Laboratory, 2006.

[131] M. Pelikan y H. Mühlenbein. *The Bivariate Marginal Distribution Algorithm.*. Advances in Soft Computing-Engineering Design and Manufacturing, 1998.

[132] M. Pelikan y H. Mühlenbein. The Bivariate Marginal Distribution Algorithm. *Advances in Soft Computing-Engineering Design and Manufacturing*, páginas 521–535, 1999.

[133] G. Rebane y J. Pearl. The recovery of causal polytrees from statistical data. En *Third Conference on Uncertainty in Artificial Intelligence*, páginas 222–228, 1987.

[134] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[135] T. Romero, P. Larrañaga, y B. Sierra. Learning Bayesian Networks In The Space Of Orderings With Estimation Of Distribution Algorithms. *IJPRAI*, tomo 18(4):607–625, 2004.

[136] R. Santana. A Markov Network based Factorized Distribution Algorithm for optimization. En *Proceedings of the 14th European Conference on Machine Learning (ECML–PKDD 2003)*, tomo 2837 de *Lecture Notes in Artificial Intelligence*, páginas 337–348. Springer–Verlag, Dubrovnik, Croatia, 2003.

[137] R. Santana. Estimation of distribution algorithms with Kikuchi approximations. *Evolutionary Computation*, tomo 13(1):67–97, 2005.

[138] R. Santana. *Modelación probabilística basada en modelos gráficos no dirigidos en Algoritmos Evolutivos con Estimación de Distribuciones*. Tesis Doctoral, Instituto de Cibernética, Matemática y Física, 2005.

[139] R. Santana. *Advances in Probabilistic Graphical Models for Optimization and Learning Applications in Protein Modelling*. Tesis Doctoral, University of the Basque Country, 2006.

[140] R. Santana, E. P. de León, y A. Ochoa. The Edge Incident Model. En *Second Symposium on Artificial Intelligence (CIMAF–99)*, páginas 352–359, 1999. Habana, Cuba.

[141] R. Santana, P. Larrañaga, y J. A. Lozano. Algoritmos de estimación de distribuciones para el problema de la determinación de la cadena lateral de una proteína. En *V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*. Puerto de la Cruz, Tenerife, 2007.

[142] R. Santana, P. Larrañaga, y J. A. Lozano. Component weighting functions for adaptive search with EDAs. En *Proceedings of the 2008 Congress on Evolutionary Computation CEC-2008*, páginas 4067–4074. IEEE Press, Hong Kong, 2008.

[143] R. Santana, P. Larrañaga, y J. A. Lozano. Protein Folding in Simplified Models with Estimation of Distribution Algorithms. *IEEE Transactions on Evolutionary Computation*, tomo 12(4):418–438, 2008.

[144] R. Santana, A. Ochoa, y M. Soto. The Mixture of Trees Factorized Distribution Algorithm. Informe Técnico ICIMAF 2000-129, Institute of Cybernetics, Mathematics and Physics, Havana, Cuba, 2000.

[145] R. Santana, A. Ochoa, y M. Soto. The Mixture of Trees Factorized Distribution Algorithm. En L. Spector, E. Goodman, A. Wu, W. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo, S. P. nad M. Garzon, y E. Burke, editores, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2001*, páginas 543–550. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[146] K. Sastry y D. E. Goldberg. On Extended Compact Genetic Algorithm. En *GECCO-2000, Late Breaking Papers, Genetic and evolutionary Computation Conference*, páginas 352–359, 2000.

[147] G. Schwarz. Estimating the dimension of a model. *Annals os Statistics*, tomo 7(2):461–464, 1978.

[148] M. Sebag y A. Ducoulombier. Extending Population-Based Incremental Learning to Continuos Search Spaces. En *Parallel Problem Solving from Nature - PPSN V*, páginas 418–427. Springer-Verlag, 1998. Berlin.

[149] C. Silverstein, S. Brin, R. Motwani, y J. Ullman. Scalable Techniques for Mining Causal Structures. *Data Mining and Knowledge Discovery*, tomo 4(2/3):163–192, 2000.

[150] M. Soto. *Un estudio sobre los algoritmos evolutivos basados en redes Bayesianas simplemente conectadas y su costo de evaluación*. Tesis Doctoral, Instituto de Cibernética, Matemática y Física, havana, Cuba, 2003. In spanish, adviser A. Ochoa.

[151] M. Soto, A. Ochoa, S. Acid, y L. M. de Campos. Introducing the Polytree Aproximation of Distribution Algorithm. En *Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, páginas 360–367, 1999. La Habana.

[152] M. Soto, A. Ochoa, J. Madera, y R. Santana. Estructuras gráficas simples para algoritmos evolutivos. En *Proceedings of the Segundo Encuentro Nacional de Computacionón ENC-99*, páginas 79–84. Pachuca, México, 1999.

[153] P. Spiessens y B. Manderick. A massively parallel genetic algorithm. En L. B. R. Belew, editor, *4th ICGA*, páginas 279–286. Morgan Kaufmann, 1991.

[154] P. Spirtes, C. Glymour, y R. Scheines. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, tomo 9:62–72, 1991.

[155] P. Spirtes, C. Glymour, y R. Scheines. *Causation, Prediction, and Search*. Springer/Verlag, first edición, 1993.

[156] P. Spirtes, C. Glymour, y R. Scheines. *Causation, Prediction, and Search*. The MIT Press, 2000.

[157] P. Spirtes y C. Meek. Learning Bayesian networks with discrete variables from data. En *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, páginas 294–299. Morgan Kaufmann, San Francisco, 1995.

[158] M. Sánchez. *Herramientas de computación distribuida de apoyo a investigaciones científicas en el área de computación evolutiva*. Proyecto Fin de Carrera, Universidad de La Habana, 2002.

[159] R. Tanese. Parallel Genetic Algorithm for Hypercube. En J. J. Grefenstette, editor, *In Proceedings of the Second International Conference on Genetic Algorithms*, páginas 177–183, 1987. Hillsdale, NJ: Lawrence Erlbaum Associates.

[160] G. Tel. *Topics in Distributed Algorithms*. Cambridge University Press, 1991.

[161] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, New York, NY, USA, 2001. ISBN 0521794838.

[162] I. Tsamardinos, C. Aliferis, y A. Statnikov. Time and sample efficient discovery of markov blankets and direct causal relations. En *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003)*, 2003.

[163] I. Tsamardinos, L. Brown, y C. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, tomo 65(1):31–78, 2006.

[164] T. S. Verma y J. Pearl. An angorithm for deciding if a set of observed independencies has a causal explanation. En *Proceedings of the Eighth international conference on uncertainty in artificial intelligence*, 1992.

[165] D. L. Whitley. An Executable Model of a Simple Genetic Algorithm. En D. L. Whitley, editor, *In Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, páginas 45–62, 1992.

[166] C. Yanover y Y. Weiss. Finding the M Most Probable Configurations Using Loopy Belief Propagation. En S. Thrun, L. Saul, y B. Schölkopf, editores, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[167] J. S. Yedidia, W. T. Freeman, y Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Informe Técnico TR-2001-35, Mitsubishi Electric Research Laboratories, 2002.

[168] J. S. Yedidia, W. T. Freeman, y Y. Weiss. Understanding belief propagation and its generalizations. Informe Técnico TR-2001-22, Mitsubishi Electric Research Laboratories, 2002.

[169] B. T. Zhang y D. Y. Cho. Evolving Neural Tress for Time Series Prediction using Bayesian Evolutionary Algorithms. En *Proceedings of the First IEEE Workshop on Combinations of Evolutionary Computation and Neural Networks*, 2000.

ANEXO A

FUNCIONES OBJETIVO

A.1. Funciones binarias

F_{OneMax}**:** Es una función separable y tiene $(n+1)$ valores diferentes de aptitud. El máximo global lo alcanza en $(1,1,\ldots,1)$ y es igual al número de variables n.

$$F_{OneMax}(\overrightarrow{x}) = \sum_{i=1}^{n} x_i \tag{A.1}$$

$F_{Plateau}$**:** Propuesta por Mühlenbein y Schlierkamp-Voosen [105] y conocido como el problema *royal road* de $3-bits$. La solución es un vector $n-dimensional$, tal que $n = 3 \cdot m$. El máximo global lo alcanza en $(1,1,\ldots,1)$ y es igual a m.

$$F_{Plateau}(\overrightarrow{x}) = \sum_{i=1}^{m} g(x_{3i-2}, x_{3i-1}, x_{3i}) \tag{A.2}$$

$$g(x_1,x_2,x_3) = \begin{cases} 1, & si\ x_1 = 1\ y\ x_2 = 1\ y\ x_3 = 1 \\ 0, & en\ otro\ caso \end{cases}$$

$F_{IsoPeak}$**:** La solución de esta función [90] está compuesta por un vector $n-dimensional$, tal que $n = 2 \cdot m$. El máximo global lo alcanza en $(1,1,0,0,\ldots,0,0)$ y es igual a m^2.

$$F_{IsoPeak}(\overrightarrow{x}) = Iso2\,(x_1,x_2) + \sum_{i=2}^{m} Iso1\,(x_{2i-1}, x_{2i}) \tag{A.3}$$

$\overrightarrow{x}$	00	01	10	11
$Iso1$	m	0	0	$m-1$
$Iso2$	0	0	0	m

$F_{Quadratic}$**:** La solución de esta función [132] está compuesta por un vector $n-dimensional$, tal que $n = 2 \times m$. El máximo global lo alcanza en $(1,1,\ldots,1)$ y es igual a m.

$$F_{Quadratic}(\overrightarrow{x}) = \sum_{i=1}^{m} g(x_{2i-1}, x_{2i}) \tag{A.4}$$

$$g(u,v) = \begin{cases} 0{,}9, & si\ u = 0\ y\ v = 0 \\ 1{,}0, & si\ u = 1\ y\ v = 1 \\ 0{,}0, & en\ otro\ caso \end{cases}$$

F_{Cuban1}**:** La solución de esta función [103] está compuesta por un vector $n-dimensional$, tal que $n = 4 \times m + 1$ y m impar. El máximo global está formado por cadenas donde se alternan sub-cadenas de la forma 10000 y 00101. La primera sub-cadena es el óptimo de la función F^5_{Cuban1}, pero la segunda toma solo el tercer mejor valor. El óptimo es difícil de alcanzar, aún cuando se utilizan técnicas de búsqueda local [103]. El valor del óptimo global para 21, 69 y 101 variables es 14.8, 47.2, y 68.8, respectivamente.

$$F_{Cuban1}(\overrightarrow{x}) = \sum_{i=1}^{m} F^5_{Cuban1}(x_{4i-3}, x_{4i-2}, x_{4i-1}, x_{4i}, x_{4i+1}) \tag{A.5}$$

$$F^5_{Cuban1}(x,y,z,v,w) = \begin{cases} 4 \cdot F^3_{Cuban1}(x,y,z), & si\ v = y\ y\ w = z \\ 0 & en\ otro\ caso \end{cases}$$

$$F^3_{Cuban1}(\overrightarrow{x}) = \begin{cases} 0{,}595 & para\ x = (0,0,0) \\ 0{,}200 & para\ x = (0,0,1) \\ 0{,}595 & para\ x = (0,1,0) \\ 0{,}100 & para\ x = (0,1,1) \\ 1{,}000 & para\ x = (1,0,0) \\ 0{,}050 & para\ x = (1,0,1) \\ 0{,}090 & para\ x = (1,1,0) \\ 0{,}150 & para\ x = (1,1,1) \end{cases}$$

F_{Muhl}**:** La solución de esta función [103] está compuesta por un vector $n-dimensional$, tal que $n = 5 \times m$. El máximo global lo alcanza en $(0,0,\ldots,0)$ y es igual a $4 \cdot m$.

$$F_{Muhl}(\overrightarrow{x}) = \sum_{i=1}^{m} F^5_{muhl}(x_{5i-4}, x_{5i-3}, x_{5i-2}, x_{5i-1}, x_{5i}) \tag{A.6}$$

$$F^5_{muhl}(x_1,x_2,x_3,x_4,x_5) = \begin{cases} 3{,}0, & para\ x = (0,0,0,0,1) \\ 2{,}0, & para\ x = (0,0,0,1,1) \\ 1{,}0, & para\ x = (0,0,1,1,1) \\ 3{,}5, & para\ x = (1,1,1,1,1) \\ 4{,}0, & para\ x = (0,0,0,0,0) \\ 0{,}0, & en\ otro\ caso \end{cases}$$

F_{Trap}**:** Este problema fue propuesto por Deb y Goldberg [37] y no es más que la concatenación de m funciones decepcionantes $trap$. La solución de esta función está compuesta por un vector $n-dimensional$, tal que $n = k \times m$. El máximo global lo alcanza en $(1,1,\ldots,1)$ y es igual a $5 \cdot m$.

$$F_{trap}(\overrightarrow{x}) = \sum_{i=1}^{m} trap(F_{OneMax}(\overrightarrow{s_i^k})) \tag{A.7}$$

$$trap(u) = \begin{cases} k, & para\ u = k \\ k-1-u, & en\ otro\ caso \end{cases}$$

donde $\overrightarrow{s_i^k} = (x_{ki-k+1}, x_{ki-k+2}, \ldots x_{ki})$ y $k = 5$.

*FirstPolytree*3**:** La función *FirstPolytree*3 (se referencia como *FP*3) [118] es una función separable que se descompone aditivamente con bloques de longitud tres. En cada bloque se evalúa la función f_3^{Poly}. La función f_3^{Poly} tiene como propiedad que su distribución de Boltzmann, con parámetro $\beta = 2$, tiene una estructura de poliárbol con los siguientes arcos: $x_1 \rightarrow x_2$ y $x_3 \rightarrow x_2$. El máximo global lo alcanza en $(0,0,1,\ldots,0,0,1)$ y es igual a $l * 1{,}074$; donde $n = 3 * l$.

$$F_{FP3}(\overrightarrow{x}) = \sum_{i=1}^{l} f_3^{Poly}(x_{3*i-2}, x_{3*i-1}, x_{3*i}) \tag{A.8}$$

x	f_3^{Poly}	x	f_3^{Poly}
000	-1.186	100	-4.391
001	1.074	101	-1.122
010	-0.469	110	-0.083
011	-0.096	111	0.553

*FirstPolytree*5**:** La función *FirstPolytree*5 (se referencia como *FP*5) [118] es una función separable que se descompone aditivamente con bloques de longitud cinco. En cada bloque se evalúa la función f_5^{Poly}. La función f_5^{Poly} tiene como propiedad que su distribución de Boltzmann, con parámetro $\beta = 2$, tiene una estructura de poliárbol con los siguientes arcos: $x_1 \rightarrow x_3$, $x_2 \rightarrow x_3$, $x_3 \rightarrow x_5$ y $x_4 \rightarrow x_5$. El máximo global lo alcanza en $(0,1,0,0,1,\ldots,0,1,0,0,1)$ y es igual a $l * 1{,}723$; donde $n = 5 * l$.

$$F_{FP5}(\overrightarrow{x}) = \sum_{i=1}^{l} f_5^{Poly}(x_{5*i-4}, x_{5*i-3}, x_{5*2}, x_{5*i-1}, x_{5*i}) \tag{A.9}$$

x	f_5^{Poly}	x	f_5^{Poly}	x	f_5^{Poly}	x	f_5^{Poly}
00000	-1.141	01000	-0.753	10000	-3.527	11000	-6.664
00001	1.334	01001	1.723	10001	-1.052	11001	-4.189
00010	-5.353	01010	-4.964	10010	-7.738	11010	-10.876
00011	-1.700	01011	-1.311	10011	-4.085	11011	-7.223
00100	0.063	01100	1.454	10100	1.002	11100	-1.133
00101	-0.815	01101	0.576	10101	0.124	11101	-2.011
00110	-0.952	01110	0.439	10110	-0.013	11110	-2.148
00111	-0.652	01111	0.739	10111	0.286	11111	-1.849

A.2. Funciones continuas

F_{Sphere}**:** Este es un conocido problema de optimización, utilizado para crear una línea base en la comparación con otros problemas o algoritmos. Las variables x_i están definidas en el intervalo $-600 \leq x_i \leq 600, i = 1,2,\ldots,n$. El mínimo global lo alcanza cuando todas las variables toman valor cero y es igual a cero.

$$F_{Sphere}(\overrightarrow{x}) = \sum_{i=1}^{n} x_i^2 \tag{A.10}$$

$F_{Griewangk}$**:** En este problema las variables x_i están definidas en el intervalo $-600 \leq x_i \leq 600$, $i = 1,2,\ldots,n$. El mínimo global lo alcanza cuando todas las variables toman valor cero y es igual a cero.

$$F_{Griewangk}(\overrightarrow{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{A.11}$$

F_{Ackley}**:** En este problema las variables x_i están definidas en el intervalo $-6{,}0 \leq x_i \leq 6{,}0, i = 1,2,\ldots,n$. El mínimo global lo alcanza cuando todas las variables toman valor cero y es igual a cero.

$$F_{Ackley}(\overrightarrow{x}) = -20 \cdot \exp\left(-0{,}2 \cdot \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(2 \cdot \pi \cdot x_i)\right) \tag{A.12}$$

F_{Water}**:** Este problema de minimización fue propuesto inicialmente por Bohachevsk y col. [18] para dos variables. Las variables x_i están definidas en el intervalo $-1{,}5 \leq x_i \leq 1{,}5, i = 1,2,\ldots,n$. El mínimo global lo alcanza cuando todas las variables toman valor cero y es igual a cero. Varios algoritmos de optimización realizan un gran número de evaluaciones en este problema al igual que algunos métodos de búsqueda especializados [64].

$$F_{Water}(\overrightarrow{x}) = \sum_{i=1}^{n/2} a \cdot x_{2i-1}^2 + b \cdot x_{2i}^2 - c \cdot \cos(\alpha \cdot x_{2i-1}) - d \cdot \cos(\gamma \cdot x_{2i}) + c + d \tag{A.13}$$

Donde $a = 1{,}0, b = 50{,}0, c = 3{,}0, d = 4{,}0, \alpha = 3 \cdot \pi$, y $\gamma = 4 \cdot \pi$.

A.3. Funciones de Boltzmann

El formalismo de las B-funciones se introduce por Ochoa y Soto [117]. Para los propósitos de esta tesis, la exposición presentada en [120] es suficiente, por lo que el lector interesado se remite al mismo para más detalles. La definición de B-función es como sigue:

Definición 9. (B-funciones): Se asume $\beta > 0$. Sea $P(x)$ una distribución de probabilidad de masa arbitraria y x_{mpc} su configuración más probable. La función paramétrica

$$BF_{P,\beta}(x) = \frac{1}{\beta}\log(\frac{P(x_{mpc})}{P(x)})$$

es llamada B-función, es no negativa, unimodal y aditiva , y tiene un mínimo en x_{mpc}. El parámetro β se conoce como la temperatura inversa de la función, mientras que $P(x)$ se conoce como su función de distribución.

Las B-funciones tienen un número importante de buenas cualidades que las hacen interesantes como banco de prueba de los algoritmos EDA [120]. Una de ellas es que para determinadas subclases de B-funciones es posible construir generadores aleatorios eficientes que generen ejemplos particulares de funciones cada vez que sean invocados. Uno de estos casos son las B-funciones con estructura de poliárbol[1], que son las que tienen un $P(x)$ que es fiel a un poliárbol. Las B-funciones tienen las siguientes propiedades:

- El mínimo de cualquier B-función es siempre cero, por lo que el criterio de parada del algoritmo es fácil de implementar y especificar.
- La computación de la configuración más probable, la cual es necesaria para la definición de la B-función, puede ser computada en tiempo polinomial para una gran clase de distribuciones [109, 166].
- La generación aleatoria de instancias de B-funciones puede realizarse eficientemente [118].
- Existe una convención de nombre para referenciar las B-funciones de forma fácil.
- No es necesario concatenar sub-funciones pequeñas para generar problemas de mayores dimensiones.

Existe una manera especial de nombrar las B-funciones [117], por ejemplo:

BF2B30s3-3445: Función binaria de 30 variables, estructura de poliárbol con un máximo de tres padres por variable. La información mutua marginal se encuentra en el intervalo $[0{,}3, 0{,}4]$, mientras que sus probabilidades univariadas están en el intervalo $[0{,}45, 0{,}55]$ (alta entropía).

BF2B100s0-0012: Función univariada (variables independientes), no tiene sentido la especificación de los valores de información mutua marginal. Sus probabilidades univariadas están en el intervalo $[0{,}15, 0{,}25]$ (baja entropía). Esta subclase de funciones se nombran B-funciones aleatorias univariadas (**RBUF**).

[1] No existe más de un camino no orientado entre cualquier para de nodos

A.4. Predicción de estructuras de proteínas

Las proteínas son cadenas o secuencias formadas por la combinación de 20 aminoácidos enlazados mediante enlaces peptidicos. En el espacio, dicha cadena adopta una estructura tridimensional, la que determina la funcionalidad biológica de la proteína. Esta estructura tiene cavidades y salientes que permiten el acoplamiento con otras proteínas para formar estructuras mas complejas, o para bloquear el funcionamiento de otras.

Estructuralmente, las proteínas pueden analizarse a diferentes escalas. Se denomina estructura primaria de una proteína, a la secuencia de aminoácidos que la componen. Estos aminoácidos se agrupan en estructuras llamadas α hélices, laminas β y loops, las cuales reciben el nombre de estructuras secundarias. Estas subestructuras se pliegan o "doblan" en el espacio tridimensional hasta alcanzar una configuración que se conoce como estructura terciaria o estado nativo. Es esta estructura tridimensional la que determina la funcionalidad biológica de la proteína [19].

Es aceptado que uno de los elementos que mas influye en la determinación de esta estructura, es el llamado efecto hidrofóbico. Los aminoácidos pueden clasificarse en hidrofílicos o hidrofóbicos según sea su comportamiento en un medio acuoso. En el proceso de plegado los aminoácidos hidrofóbicos tienden a agruparse en el centro de la molécula formando una especie de coraza interna, mientras que los hidrofílicos tienden a quedar expuestos al solvente.

A partir de esto podemos definir el problema de la predicción de la estructura de una proteína como la búsqueda, a partir de la secuencia de aminoácidos, de la estructura tridimensional correspondiente.

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

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

Los modelos en retículos utilizan las coordenadas internas para modelizar las estructuras terciarias de las proteínas, dada la posición del aminoácido i, existen δ valores para representar la posición del $i+1$ dependiendo del retículo utilizado (bidimensional, tridimensional o de diamante).

El modelo mas simple de PSP en retículos, llamado **modelo de Dill** [38]. Solo considera 2 tipos de aminoácidos, donde cada tipo representa si es hidrofóbico (representado con una H) o hidrofílico (representado con una P). Una proteína entonces, se modeliza como secuencia $s \in \{H,P\}^+$. En la figura A.1 se muestra un ejemplo de configuración de una proteína en el modelo HP.

	H	P
H	-1	0
P	0	0

	H	P
H	-3	-1
P	-1	0

Tabla A.1: Dos posibles matrices de interacción $\varepsilon_{i,j}$

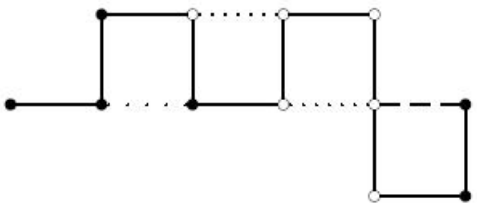

Figura A.1: Una posible configuración para la secuencia $HHHPHPPPPPH$ para el modelo HP en un retículo bidimensional. La configuración muestra un contacto HH, uno HP y dos PP

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

$$E(S,X) = \sum_{i}^{n} \sum_{j>i+1}^{n} \varepsilon_{i,j} \star \triangle(x_i, x_j) \tag{A.14}$$

donde $\triangle(x_i, x_j) = 1$ si x_i y x_j son adyacentes en el retículo y no consecutivos en la cadena y 0 en caso contrario. El término $\varepsilon_{i,j} = \varepsilon(s_i, s_j)$ es el valor de la fila i, columna j en la matriz de interacción ε. En la tabla A.1 se muestran dos matrices de interacción posibles.

Con estos elementos se establece que resolver PSP en este modelo es equivalente a minimizar la función de energía $E(S,X)$ (ecuación A.14).

Las estructuras en estos modelos se representan utilizando el sistema de coordenadas internas, de las cuales aparecen dos variaciones: la codificación mediante coordenadas absolutas y coordenadas relativas, en nuestro caso utilizamos la ultima codificación.

ANEXO B

PUBLICACIÓN DE LOS RESULTADOS

Publicaciones:

- A. Ochoa, M. Soto, R. Santana, **J. Madera** y N. Jorge. The Factorized Distribution Algorithm and The Juncction Tree. Proceedings of the Second Symposium on Artificial Itelligence. Adaptative Systems, 1999.
- **J. Madera** y A. Ochoa. Influencia de la política de migración en el Algoritmo de Distribución Marginal Univariado distribuido. Memorias de Primer Evento Internacional de Matemática Educativa e Informática. Camagüey, Noviembre 2002. ISBN 959-16-0179-4.
- **J. Madera**. Utilización de las redes de computadoras como máquinas paralelas. Memorias del II Congreso Internacional de Telemática. La Habana, Noviembre 2004.
- **J. Madera** y L. Duarte. Implementación de un algoritmo EDA con evaluación distribuida. Memorias de III Seminario Internacional de Matemática, Física e Informática Educativa. Camagüey, Noviembre 2004. ISBN 959-16-0294-0.
- **J. Madera**, E. Alba, A. Ochoa, Parallel estimation of distribution algorithms, en: E. Alba (Ed.), Parallel Metaheuristics: A New Class of Algorithms, John Wiley & Sons, Inc., 2005.
- **J. Madera**, E. Alba, A. Ochoa, A parallel island model for estimation of distribution algorithms, en: J.A. Lozano, P. Larrañaga, I. Inza y E. Bengoetxea (Eds.), Towards a new evolutionary computation. Advances on estimation of distribution algorithms, Springer, 2006.
- **J. Madera**, B. Dorronsoro. Estimation of Distribution Algorithms. E. Alba y R. Marti (eds.), en: Metaheuristic Procedures for training Neural Networks, Springer, 2006.
- E. Alba, **J. Madera**, B. Dorronsoro, A. Ochoa y M. Soto. Theory and practice of cellular UMDA for discrete Optimization. Parallel Problem Solving from Nature, Granada, 2006.
- **J. Madera**, E. Alba, G. Luque. Performance Evaluation of the Parallel Polytree Approximation Distribution Algorithm on Three Network Technologies. XII Congreso Argentino de Ciencias de la Computación (CACIC 2006). Universidad Nacional de San Luis, Argentina, 2006.
- **J. Madera** y L. Duarte. Biblioteca para el Rápido Desarrollo de Algoritmos Evolutivos que Estiman Distribuciones. Memorias de la II Conferencia Científica UCIENCIA. La Habana, 2006. ISBN 959-16-0463-7.
- **J. Madera**, M. Díaz, A. Ochoa y M. Soto. Implementación Paralela de un Algoritmo Evolutivo con Estimación de la Distribución Basado en Redes Simplemente Conectadas. Memorias de la

VII Conferencia Científica Internacional de la UNICA. Ciencias de la Computación. Ciego de Ávila, 2006. ISBN 959-16-0473-4.

- **J. Madera**, A. Ochoa. Un EDA basado en aprendizaje MAX-MIN con escalador de colinas. Instituto de Cibernética, Matemática y Física (ICIMAF), 2006. ISSN 0138-8916.
- **J. Madera**, A. Ochoa. Una versión paralela del algoritmo MMHCEDA. Instituto de Cibernética, Matemática y Física (ICIMAF), 2006.
  ISSN 0138-8916.
- **J. Madera**, E. Alba, G. Luque. Performance Evaluation of the Parallel Polytree Approximation Distribution Algorithm on Three Network Technologies. Revista Iberoamericana de Inteligencia Artificial, vol. 11 (35), 2007.

En el marco de esta investigación se realizó la tesis de maestría:

J. Madera (2008). Algoritmos Evolutivos con Estimación de Distribuciones basados en Pruebas de Independencia, Tesis de Maestría, Universidad de Camagüey, Camagüey, Cuba.

Los resultados de la tesis se han discutido en los seminarios de algunos grupos de investigación en Sistemas Adaptativos e Inteligencia Artificial de las siguientes instituciones:

- Universidad de Málaga, España.
- Universidad de las Islas Baleares, España.
- Universidad APEC, República Dominicana.
- Instituto Superior de Ciencias Médicas de Camagüey, Cuba.