ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS INFORMÁTICOS
UNIVERSIDAD POLITÉCNICA DE MADRID

TESIS DE MÁSTER
MÁSTER UNIVERSITARIO EN
INTELIGENCIA ARTIFICIAL

COMPUTACIÓN EVOLUTIVA DE BOSQUES DE EXPANSIÓN MÍNIMOS CON RESTRICCIÓN DE GRADO Y DE ROL

AUTOR : Laura Antón Sánchez
TUTORES : Concha Bielza Lozoya
Pedro Larrañaga Mugica

Julio, 2015

Reconocimientos

A mis supervisores, Concha Bielza y Pedro Larrañaga, por su apoyo y valiosos consejos durante toda la realización de esta tesis fin de máster.
Al Ministerio de Economía y Competitividad a través del proyecto Cajal Blue Brain (C080020-09).
A mis compañeros del Computational Intelligence Group por su compañía y ayuda.
A mi familia y amigos por su apoyo incondicional.

Resumen

Encontrar el árbol de expansión mínimo con restricción de grado de un grafo (DCMST por sus siglas en inglés) es un problema NP-complejo ampliamente estudiado. Una de sus aplicaciones más importantes es el diseño de redes. Aquí nosotros tratamos una nueva variante del problema DCMST, que consiste en encontrar el árbol de expansión mínimo no sólo con restricciones de grado, sino también con restricciones de rol (DRCMST), es decir, añadimos restricciones para restringir el rol que los nodos tienen en el árbol. Estos roles pueden ser nodo raíz, nodo intermedio o nodo hoja. Por otra parte, no limitamos el número de nodos raíz a uno, por lo que, en general, construiremos bosques de DRCMSTs. El modelado en los problemas de diseño de redes puede beneficiarse de la posibilidad de generar más de un árbol y determinar el rol de los nodos en la red.

Proponemos una nueva representación basada en permutaciones para codificar los bosques de DRCMSTs. En esta nueva representación, una permutación codifica simultáneamente todos los árboles que se construirán. Nosotros simulamos una amplia variedad de problemas DRCMST que optimizamos utilizando ocho algoritmos de computación evolutiva diferentes que codifican los individuos de la población utilizando la representación propuesta. Los algoritmos que utilizamos son: algoritmo de estimación de distribuciones (EDA), algoritmo genético generacional (gGA), algoritmo genético de estado estacionario (ssGA), estrategia evolutiva basada en la matriz de covarianzas (CMAES), evolución diferencial (DE), estrategia evolutiva elitista (ElitistES), estrategia evolutiva no elitista (NonElitistES) y optimización por enjambre de partículas (PSO). Los mejores resultados fueron para el algoritmo de estimación de distribuciones utilizado y ambos tipos de algoritmos genéticos, aunque los algoritmos genéticos fueron significativamente más rápidos.

Abstract

Finding the degree-constrained minimum spanning tree (DCMST) of a graph is a widely studied NP-hard problem. One of its most important applications is network design. Here we deal with a new variant of the DCMST problem, which consists of finding not only the degree- but also the role-constrained minimum spanning tree (DRCMST), i.e., we add constraints to restrict the role of the nodes in the tree to root, intermediate or leaf node. Furthermore, we do not limit the number of root nodes to one, thereby, generally, building a forest of DRCMSTs. The modeling of network design problems can benefit from the possibility of generating more than one tree and determining the role of the nodes in the network.

We propose a novel permutation-based representation to encode the forest of DRCMSTs. In this new representation, one permutation simultaneously encodes all the trees to be built. We simulate a wide variety of DRCMST problems which we optimize using eight different evolutionary computation algorithms encoding individuals of the population using the proposed representation. The algorithms we use are: estimation of distribution algorithm (EDA), generational genetic algorithm (gGA), steady-state genetic algorithm (ssGA), covariance matrix adaptation evolution strategy (CMAES), differential evolution (DE), elitist evolution strategy (ElististES), non-elitist evolution strategy (NonElististES) and particle swarm optimization (PSO). The best results are for the estimation of distribution algorithm and both types of genetic algorithms, although the genetic algorithms are significantly faster.

Contenidos

Lista de imágenes

LISTA DE IMÁGENES

Lista de tablas

LISTA DE TABLAS

1

Introducción

Un árbol de expansión es una estructura topológica básica en problemas de diseño de redes, tales como los sistemas de transporte, telecomunicaciones y sistemas de distribución. Existen algoritmos clásicos bien conocidos para la construcción de un árbol de expansión mínimo (MST por sus siglas en inglés, *minimum spanning tree*) (Kruskal, 1956; Prim, 1957). En la práctica una representación más realista para el diseño de una red es un árbol de expansión mínimo con restricción de grado (conocido también como *degree-constrained minimum spanning tree* o DCMST), es decir, un MST con restricciones sobre el número de aristas incidentes a cada nodo. La Figura 1.1(a) muestra un ejemplo del MST de un grafo con 7 nodos. El coste de cada conexión está indicado sobre las aristas que unen los nodos. La Figura 1.1(b) muestra el DCMST del mismo grafo, donde cada nodo puede tener un máximo de tres aristas incidentes.

El problema DCMST se puede aplicar en un sistema de transporte a través de, por ejemplo, cableado, tubos o canales, donde la longitud de las conexiones de m nodos debe ser mínima. La capacidad de cada nodo impone una restricción sobre el número de aristas que pueden conectarse a ese nodo. En las redes de comunicación, la restricción de grado limita la vulnerabilidad de la red si un nodo falla. El problema DCMST también podría aplicarse al diseño de una red de ordenadores o una red de carreteras con un número máximo de carreteras en un cruce (Krishnamoorthy et al., 2001).

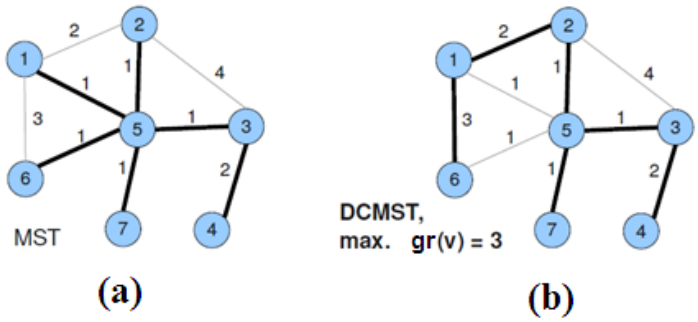


Figura 1.1: Ejemplo de MST (a) y DCMST (b) de un grafo. Sobre cada arista se indica su coste. En (b) se fija el grado máximo de cada nodo en 3.

El problema DCMST es NP-complejo (puede ser demostrado por reducción al problema del camino hamiltoniano, Garey y Johnson (1979)). El problema de encontrar el DCMST de un grafo y, en particular, de encontrar una buena representación del árbol, ha sido ampliamente estudiado en la literatura. Por ejemplo, Knowles et al. (2000) introducen el *randomized primal method*, un algoritmo novedoso de construcción del árbol con técnicas de búsqueda estocásticas iterativas. Este método construye árboles con restricción de grado de bajo coste. Krishnamoorthy et al. (2001) comparan tres heurísticas (recocido simulado, un algoritmo genético y un método basado en el espacio de búsqueda del problema) y dos algoritmos exactos (relajación lagrangiana y ramificación y poda) para el problema DCMST. Además, proponen representaciones alternativas de los árboles para facilitar las búsquedas de vecinos al algoritmo genético. Raidl y Julstrom (2003) proponen representar los árboles de expansión para problemas de diseño de redes directamente como conjuntos de sus ejes. Los autores demuestran la utilidad de su codificación para el problema DCMST. Soak et al. (2004) desarrollan otro método efectivo de codificación de un árbol para el uso de métodos de optimización de caja-negra.

Las representaciones anteriores se basan en la construcción de un solo árbol. Algunos estudios más recientes consideran la construcción de un bosque. Esta extensión no es sencilla. En Delbem et al. (2004), la representación propuesta para el bosque, llamada *node-depth encoding*, está compuesta por la unión de la codificación de todos los árboles del bosque. La unión se realiza utilizando una matriz de punteros, donde cada puntero indica un árbol y cada árbol consta de listas que contienen los nodos del árbol y sus profundidades. El enfoque propuesto se evalúa para el problema DCMST. Algunos años más tarde, Delbem et al. (2012) proponen una nueva estructura de datos para generar y manipular un conjunto de bosques de

expansión y llaman a esta representación *node-depth-degree encoding*. Esta estructura mejora el tiempo de ejecución promedio de su anterior codificación *node-depth* (el bosque está de nuevo formado por la unión de los árboles). Trabajando también con un grupo de árboles, Czajko y Wojciechowski (2009) toman un enfoque diferente. Ellos formulan el problema de encontrar el *hop- and degree-constrained minimum spanning forest* con minimización del número de árboles (este problema se define como parte del diseño de la topología de una red de acceso).

Aquí definimos otra variante del problema DCMST, que llamamos DRCMST (*degree- and role-constrained minimum spanning tree*). Un DRCMST es un DCMST donde determinamos a priori el rol de los nodos en el árbol de acuerdo a tres alternativas: nodo raíz, nodo intermedio y nodo hoja. Restringir el rol de los nodos puede ser útil en diseño de redes. En redes de ordenadores (Fig 1.2(a)), por ejemplo, el servicio tiene que llegar a los nodos hoja, y estos nodos son claramente diferentes a los procesadores centrales (que tiene un número fijo de puertos). En una red de este tipo el coste podría estar asociado con las distancias entre los nodos o con los costes del material necesario para conectar los nodos. Un DRCMST también podría ser útil en una red de negocios (Fig 1.2(b)), por ejemplo, para el diseño de la estructura de personal de un proyecto, donde diferenciaríamos entre el director del proyecto (raíz), mandos intermedios (nodos intermedios) y el resto del equipo que no están a cargo de otro personal (nodos hoja). El coste de este problema podría estar asociado con las preferencias de los miembros del equipo hacia los directores de proyecto.

El problema DRCMST es también NP-complejo porque contiene el caso particular en que fijamos un nodo raíz y un nodo hoja con la restricción de que el grado de cada nodo tiene que ser menor o igual a dos. Esto es equivalente al problema del camino hamiltoniano más corto entre dos nodos. Además, permitimos construir un bosque en lugar de un solo árbol, es decir, no limitamos el número de nodos raíz a uno, por lo que podemos resolver problemas más complejos (por ejemplo, podríamos diseñar varias redes de ordenadores y varias redes de negocios, considerando al mismo tiempo varios procesadores centrales y varios directores de proyecto en los ejemplos anteriores).

En este trabajo, presentamos una nueva representación basada en permutaciones para la construcción de bosques de DRCMSTs. Una permutación codifica simultáneamente todos los DRCMSTs en el bosque. Debido a la complejidad del problema, nosotros abordamos una amplia variedad de problemas DRCMST utilizando técnicas de computación evolutiva. Los individuos de las poblaciones están

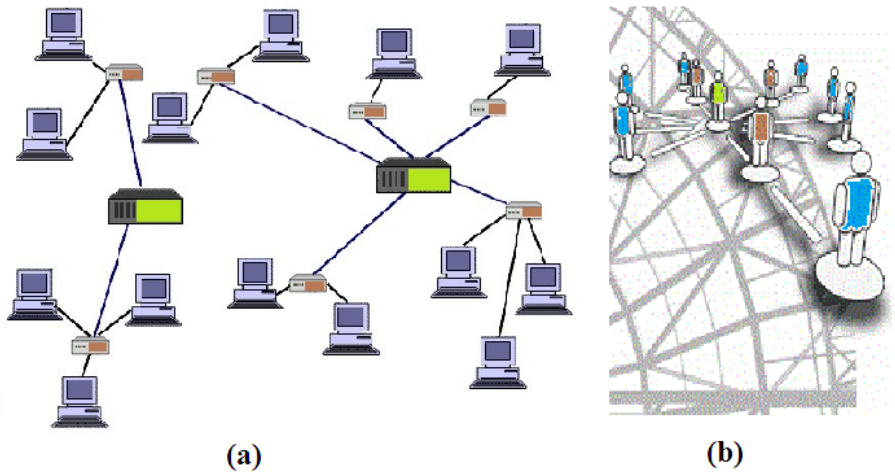


Figura 1.2: Ejemplos de aplicación del problema DRCMST. Los nodos raíz se muestran en verde, los nodos intermedios en marrón y los nodos hoja en azul. (a) Diseño de una red de ordenadores. Los ordenadores de la red (nodos hoja) están conectados a los procesadores centrales (nodos raíz) a través de los encaminadores (nodos intermedios). (b) Diseño de la estructura de personal de un proyecto. El director del proyecto (en verde) está a cargo de mandos intermedios (marrón) que a su vez están a cargo de otro personal (en azul). Vemos como el director también puede dirigir directamente a personal que no son mandos intermedios, es decir, estar conectado directamente con nodos hoja. En el problema DRCMST se fijan los roles de los nodos, se minimiza el coste total de las conexiones y se limita el grado máximo de los nodos (número de puertos en el caso de la red de ordenadores y número máximo permitido de personas a cargo en el caso de la red de personal).

codificados con la representación propuesta.

La organización de este trabajo es la siguiente. El Capítulo 2 describe formalmente los problemas DCMST y DRCMST. El Capítulo 3 introduce la representación propuesta basada en permutaciones para codificar los bosques de DRCMSTs. Esta representación se utiliza para aproximar una amplia variedad de problemas DRCMST utilizando los ocho algoritmos de computación evolutiva descritos en el Capítulo 4. En el Capítulo 5 se presenta el software utilizado e implementado para comparar el desempeño de las diferentes técnicas de computación evolutiva sobre el problema de encontrar bosques de DRCMSTs. El Capítulo 6 detalla las características de los 60 problemas de prueba simulados utilizados en el estudio. En el Capítulo 7 se analizan los resultados y se comparan los algoritmos. Por último, en el Capítulo 8 se proporcionan algunas conclusiones.

2

Definición del problema

Un DCMST es un árbol de expansión, donde se asume que hay una restricción de grado en cada nodo de tal manera que, en el nodo v, su grado $\mathrm{gr}(v)$ (es decir, su número de aristas incidentes) es a lo sumo un valor dado $g_v \in \mathbb{N}$ y el coste total de los ejes del árbol es mínimo. Formalmente, sea $G = (V, E)$ un grafo no dirigido completo con un conjunto de vértices (nodos) V y un conjunto de ejes (aristas) E. Un árbol de expansión de G es un subgrafo $A = (V, E_A)$, $E_A \subset E$ que contiene todos los vértices en V, conectados con exactamente $|V| - 1$ ejes. Sea $c_{uv} \geq 0$ el coste de cada eje $(u, v) \in E$, $u, v \in V$. El problema DCMST consiste en encontrar un árbol de expansión mínimo $A^* = (V, E_{A^*}), E_{A^*} \subset E$ tal que

$$A^* = \operatorname*{argmín}_{A} \sum_{(u,v)\in E_A} c_{uv},$$

sujeto a

$$\mathrm{gr}(v) \leq g_v \quad \text{para todo } v \in V.$$

Un DRCMST es un DCMST donde el rol de los nodos en el árbol se determina a priori (por el usuario/experto). En un problema DRCMST, definimos tres subconjuntos de nodos R, I y H para los nodos raíz, nodos intermedios y nodos hoja, respectivamente, donde cada nodo $v \in V$ debe pertenecer a uno y sólo uno de los subconjuntos, $V = R \cup I \cup H$, $R \cap I = R \cap H = I \cap H = \emptyset$, $R \neq \emptyset$ y $H \neq \emptyset$. Nótese que se deben cumplir las siguientes condiciones: $g_v \geq 1$ $\forall v \in R$, $g_v \geq 2$ $\forall v \in I$ y $g_v = 1$ $\forall v \in H$. Si seleccionamos un sólo nodo como nodo raíz ($|R| = 1$), construiremos un solo árbol. Con un número de raíces mayor, construiremos un bosque de DRCMSTs.

Dado un grafo completo no dirigido $G = (V, E)$ definido como antes, un bosque de G es un subgrafo $B = (V, E_B)$, $E_B \subset E$ que contiene todos los vértices en V y

se compone de un árbol de expansión en cada componente conexa de B. Dada una definición de las restricciones de grado y de los subconjuntos R, I y H que satisfaga los requisitos especificados en el párrafo anterior, el problema DRCMST consiste en encontrar un bosque mínimo $B^* = (V, E_{B^*}), E_{B^*} \subset E$ con $|R|$ componentes conexas tales que

$$B^* = \underset{B}{\text{argmín}} \sum_{(u,v)\in E_B} c_{uv}, \tag{2.1}$$

sujeto a

$$\begin{gathered} \text{gr}(v) \leq g_v \ \text{ para todo } v \in V \\ \text{rol}(v) = raiz \text{ para todo } v \in R \\ \text{rol}(v) = intermedio \text{ para todo } v \in I \\ \text{rol}(v) = hoja \text{ para todo } v \in H, \end{gathered}$$

donde $\text{rol}(v) \in \{raiz, intermedio, hoja\}$ indica el rol del nodo v en el bosque.

Proposición 1. Dado un grafo completo no dirigido $G = (V, E)$, los subconjuntos de nodos R, I y H tal que $V = R \cup I \cup H$, $R \cap I = R \cap H = I \cap H = \emptyset$, $R \neq \emptyset$ y $H \neq \emptyset$, y una restricción de grado por cada $v \in V$ satisfaciendo que $g_v \geq 1\ \forall v \in R$, $g_v \geq 2\ \forall v \in I$ y $g_v = 1\ \forall v \in H$, el problema DRCMST es factible si y sólo si

$$\sum_{v\in R} g_v + \sum_{v\in I} g_v - |I| \geq |I| + |H|. \tag{2.2}$$

Demostración.

$\Rightarrow$

Supongamos que el problema es factible, es decir, el bosque de G consiste en $|R|$ árboles. Vamos a demostrar que la Ecuación (2.2) se cumple para $\text{gr}(v)$, $v \in V$ y entonces también se cumplirá para $g_v, v \in V$, porque $g_v \geq \text{gr}(v)$, $\forall v \in V$. Entonces,

$$\sum_{v\in R}\text{gr}(v) + \sum_{v\in I}\text{gr}(v) - |I| \geq |I| + |H| \Longleftrightarrow \sum_{v\in R}\text{gr}(v) + \sum_{v\in I}\text{gr}(v) \geq 2|I| + |H|$$

Sumando $\sum_{v\in H}\text{gr}(v)$ a ambos lados de la Ecuación (2.2):

$$\sum_{v\in R}\text{gr}(v) + \sum_{v\in I}\text{gr}(v) + \sum_{v\in H}\text{gr}(v) \geq 2|I| + |H| + \sum_{v\in H}\text{gr}(v)$$

Se sabe que $\sum_{v\in V}\text{gr}(v) = 2|E|$ se cumple para todo grafo $G = (V, E)$. Además, debido a que $g_v = \text{gr}(v) = 1\ \forall v \in H$, $\sum_{v\in H}\text{gr}(v) = |H|$ y tenemos que

$$\sum_{v \in V} \text{gr}(v) \geq 2|I| + |H| + |H| \iff 2|E| \geq 2|I| + 2|H| \iff |E| \geq |I| + |H|$$

Puesto que $|E_A| = |V_A| - 1$ se cumple para todo árbol A y el bosque tiene $|R|$ árboles $\Rightarrow |E| = |V| - |R|$. Entonces, $|V| - |R| \geq |I| + |H|$ y se convierte en igualdad porque $|V| = |R| + |I| + |H|$. Por lo tanto, debido a que la Ecuación (2.2) se cumple para $\text{gr}(v)$, $v \in V$, también es cierto para $g_v, v \in V$.

$\Leftarrow$

Supongamos que la Ecuación (2.2) se cumple. Veamos que el problema es factible, es decir, que podemos construir $|R|$ árboles.

Sumamos $\sum_{v \in H} g_v$ a ambos lados de la Ecuación (2.2) y razonando como en el caso anterior tenemos que:

$$\sum_{v \in R} g_v + \sum_{v \in I} g_v + \sum_{v \in L} g_v \geq 2|I| + |H| + \sum_{v \in H} g_v \iff \sum_{v \in V} g_v \geq 2|I| + |H| + |H| \iff 2|E| \geq 2|I| + 2|H| \iff |E| \geq |I| + |H|$$

Puesto que $|V| = |R| + |I| + |H| \Rightarrow |I| + |H| = |V| - |R|$, tenemos que $|E| \geq |V| - |R|$. Para construir $|R|$ árboles es necesario que $|E| = |V| - |R|$, por lo tanto, el problema es factible. ■

En otras palabras, el problema DRCMST es factible si y sólo si el número máximo permitido de ejes de "salida" (lado izquierdo de la Ecuación (2.2)) es mayor o igual que el número de ejes de "entrada" (lado derecho de la Ecuación (2.2)). Ver Figura 2.1 para un contraejemplo. Nótese que estamos trabajando con grafos no dirigidos por lo que realmente no existen ejes de entrada o salida. Sin embargo, al establecer los nodos raíz, la estructura de árbol tiene dirección implícitamente ya se considera que que las raíces (hojas) son el origen (final) de un árbol.

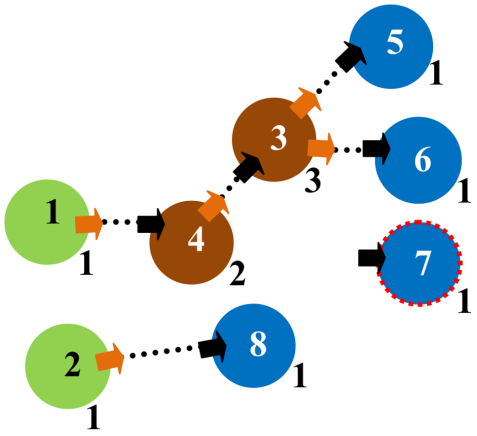


Figura 2.1: Ejemplo de un problema DRCMST infactible. El grado máximo permitido g_v se muestra a la derecha de cada nodo. Los nodos raíz se muestran en verde, los nodos intermedios en marrón y los nodos hoja en azul. Puesto que $|R| = 2$, $|I| = 2$ y $|H| = 4$, necesitamos seis "entradas "(flechas negras): dos para los nodos intermedios y cuatro para los nodos hoja. Sin embargo, sólo disponemos de cinco posibles "salidas "(flechas naranjas): dos de los nodos raíz y tres de los nodos intermedios. En este ejemplo, el nodo número 7 no se puede conectar a ninguno de los dos árboles de este bosque. Este ejemplo no satisface la Ecuación (2.2): $2 + 5 - 2 \not\geq 2 + 4$.

3

Representación del problema

Nosotros proponemos codificar el problema DRCMST utilizando una representación basada en permutaciones. Una permutación es un vector $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_n)$ de los índices $1, ..., n$ tal que $\sigma_k \neq \sigma_s$ para todo $k \neq s$. Decimos que el índice s está en la posición k en $\boldsymbol{\sigma}$ cuando $\sigma_k = s$.

En un bosque codificado por la representación propuesta, todos los nodos tienen un grado $\mathrm{gr}(v)$ igual a su grado máximo permitido g_v. Para verificar esta restricción, añadimos un nuevo tipo de nodos que llamamos nodos ficticios, ver Figura 3.1. Añadimos tantos nodos ficticios como sean necesarios para hacer $\mathrm{gr}(v) = g_v, \forall v \in V$. Sea F el subconjunto de nodos ficticios, en un bosque de DRCMSTs codificado por nuestra representación, la Ecuación (2.2) se convierte en una igualdad:

$$\sum_{v \in R} g_v + \sum_{v \in I} g_v - |I| = |I| + |H| + |F|, \tag{3.1}$$

y, por lo tanto, el número de nodos ficticios que se añaden es

$$|F| = \sum_{v \in R} g_v + \sum_{v \in I} g_v - 2|I| - |H|. \tag{3.2}$$

Los nodos ficticios se añaden sólo por requisitos de la representación y son todos nodos hoja, por lo que su grado es siempre igual a 1. El coste de todo eje que llega a un nodo ficticio es cero. Entonces, $m = |V| + |F| = |R| + |I| + |H| + |F|$ es el número total de nodos en el bosque codificado.

En nuestra representación, cada índice de la permutación denota una conexión entre dos nodos, es decir, cada posición de la permutación representa un eje en el bosque. Puesto que $|E_A| = |V_A| - 1$ se cumple para todo árbol A y nosotros codi-

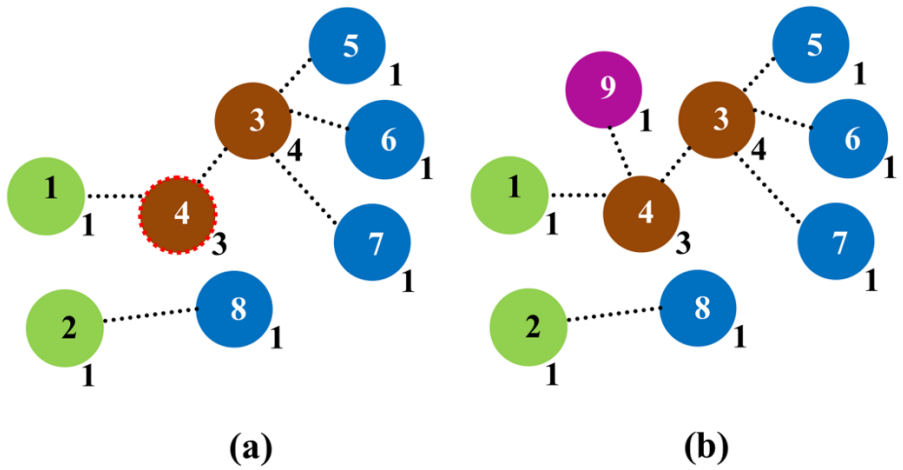


Figura 3.1: Ejemplo de un bosque de DRCMSTs con dos árboles. El grado máximo permitido g_v se muestra a la derecha de cada nodo. En (a) el grado gr(v) de todos los nodos es igual a su grado máximo permitido g_v, excepto para el nodo número 4 donde gr(4)=2 y g_4=3. Para codificar este bosque con nuestra representación basada en permutaciones, añadimos un nodo ficticio, el nodo 9, conectado al nodo 4. Los bosques (a) y (b) son equivalentes porque los nodos ficticios se añaden sólo a efectos de representación y no afectan al cálculo de los costes de los árboles.

ficamos $|R|$ árboles en una permutación, la longitud de la permutación n (número total de ejes en el bosque codificado) se puede obtener como $n = m - |R|$.

La longitud de la permutación también se puede obtener usando la Ecuación (3.1):

$$n = \sum_{v \in R} g_v + \sum_{v \in I} g_v - |I| = |I| + |H| + |F|. \tag{3.1}$$

Para averiguar qué nodos están conectados por los ejes representados en cada posición de la permutación, necesitamos dos matrices auxiliares, *padres* e *hijos*, ambas de dimensiones 1xn. Estas matrices se mantienen inalteradas para todas las permutaciones de un mismo problema. La matriz auxiliar *padres* representa las "salidas"de los ejes del bosque, es decir, de qué nodos salen los ejes del bosque. Puesto que cada nodo raíz tiene g_v "salidas", y cada nodo intermedio tiene $(g_v - 1)$ "salidas", cada nodo raíz aparece g_v veces para todo $v \in R$ y cada nodo intermedio aparece $(g_v - 1)$ veces para todo $v \in I$ en *padres*. La matriz auxiliar *hijos* representa las "entradas"de los ejes, es decir, a qué nodos llegan los ejes del bosque. Todos los nodos intermedios, nodos hoja y nodos ficticios tienen una "entrada", por lo tanto la matriz *hijos* incluye todos los nodos de $v \in I \cup H \cup F$ una vez. Con estas matrices, nuestra permutación es tal que $\sigma_k = s$ representa que el nodo $padres_s$ en

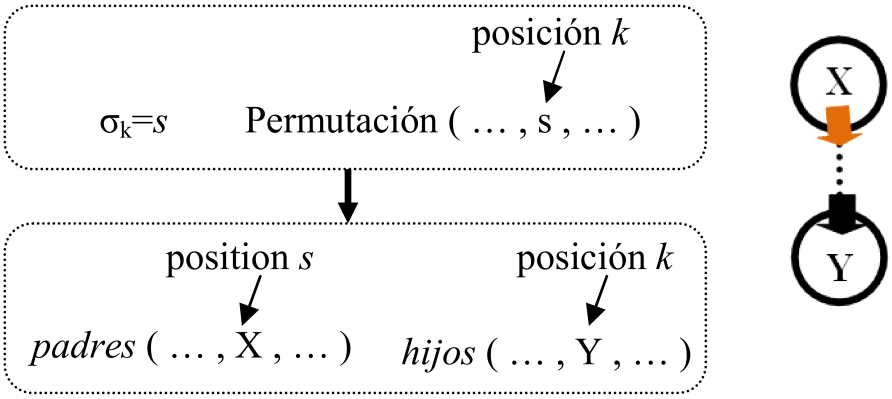


Figura 3.2: Decodificación de la representación propuesta basada en permutaciones. $\sigma_k = s$ representa que, en el bosque, el nodo $padres_s$ (nodo X) es el padre del nodo $hijos_k$ (nodo Y).

el bosque (nótese que utilizamos el subíndice para indicar el elemento en la posición s de la única fila que tiene la matriz auxiliar) es el padre del nodo $hijos_k$, véase la Figura 3.2. Una versión simplificada de esta novedosa representación considerando sólo árboles binarios se introdujo en Anton-Sanchez et al. (2013).

Para ilustrar esta representación, considérese el ejemplo de la Figura 3.3. La Figura 3.3(a) muestra los $|V| = 10$ nodos de un grafo de ejemplo $G = (V, E)$. El grado máximo permitido g_v se muestra en el lado derecho de cada nodo $v \in V$. Los nodos seleccionados como nodos raíz se muestran en color verde, los nodos intermedios en color marrón y los nodo hoja aparecen en azul. Tenemos que $|R| = 2$, $|I| = 3$ y $|H| = 5$. Este problema es factible porque satisface la Ecuación (2.2). A continuación, comprobamos si es necesario añadir algún nodo ficticio para resolver el problema haciendo uso de la Ecuación (3.2):

$$|F| = \sum_{v \in R} g_v + \sum_{v \in I} g_v - 2|I| - |H| = 3 + 9 - 2 \cdot 3 - 5 = 1,$$

es decir, tenemos que añadir un nodo ficticio. La Figura 3.3(b) muestra los nodos numerados y el nodo ficticio añadido (nodo número 11 en rosa). Entonces, un bosque en el ejemplo tendrá dos árboles ($|R| = 2$) con $m = |V| + |F| = 10 + 1 = 11$ nodos y estará representado por permutaciones de longitud $n = m - |R| = 11 - 2 = 9$.

Construimos las matrices auxiliares *padres* e *hijos* ambas necesarias para decodificar las permutaciones. Como se ha indicado, añadimos cada nodo raíz g_v veces ($v \in R$) y cada nodo intermedio ($g_v - 1$) veces ($v \in I$) a la matriz *padres*. Una posible matriz auxiliar *padres* se muestra en la Figura 3.3(c). Una posible matriz auxiliar

hijos, incluyendo los nodos intermedios, los nodos hoja y los nodos ficticios una vez, se muestra en la Figura 3.3(d). Nótese que las matrices auxiliares *padres* e *hijos* deben establecerse antes de empezar a resolver el problema, ya que determinan que solución al problema representa cada permutación. El orden de los nodos en estas matrices es irrelevante.

La Figura 3.3(e) representa la permutación (6,1,2,4,5,8,7,9,3) que sería un individuo (bosque) válido utilizando las matrices auxiliares *padres* e *hijos* definidas, es decir, $padres_6$ (nodo 4) es el padre de $hijos_1$ (nodo 3), $padres_1$ (nodo 1) es el padre de $hijos_2$ (nodo 4) y así sucesivamente hasta la última posición de la permutación, que indica que $padres_3$ (nodo 2) es el padre de $hijos_9$ (nodo 11).

Nuestra representación basada en permutaciones garantiza implícitamente que todas las restricciones del problema expresado por la Ecuación (2.1) se cumplen en el bosque codificado. Sin embargo, las permutaciones que codifican algún ciclo representan bosques no válidos. Por ejemplo, la Figura 3.3(f) representa a un individuo inválido (permutación (1,**8**,**6**,4,5,7,9,2,3)) ya que contiene un ciclo (en rojo) entre los nodos 4 y 5. La segunda posición de la permutación indica que $padres_8$ (nodo 5) es el padre de $hijos_2$ (nodo 4) y la siguiente posición indica que $padres_6$ (nodo 4) es el padre de $hijos_3$ (nodo 5). Como demostramos a continuación, podemos asegurar que las permutaciones cuyas representaciones no contienen ningún ciclo son bosques correctos, es decir, representan el número requerido de árboles $|R| = m - n$.

Proposición 2. Si un grafo $G = (V, E)$ tiene m nodos, n ejes y no tiene ciclos, entonces es un bosque compuesto por $(m - n)$ árboles.

Demostración. Supongamos que tenemos c componentes conexas en G y sea n_k el número de ejes y m_k el número de nodos en la componente k-ésima,$k = 1, ..., c$. Dado que no existen ciclos, estas componentes conexas son árboles, y para cada una de ellas se cumple que:

$n_k = m_k - 1 \quad \forall k, k = 1, ..., c.$

Entonces,

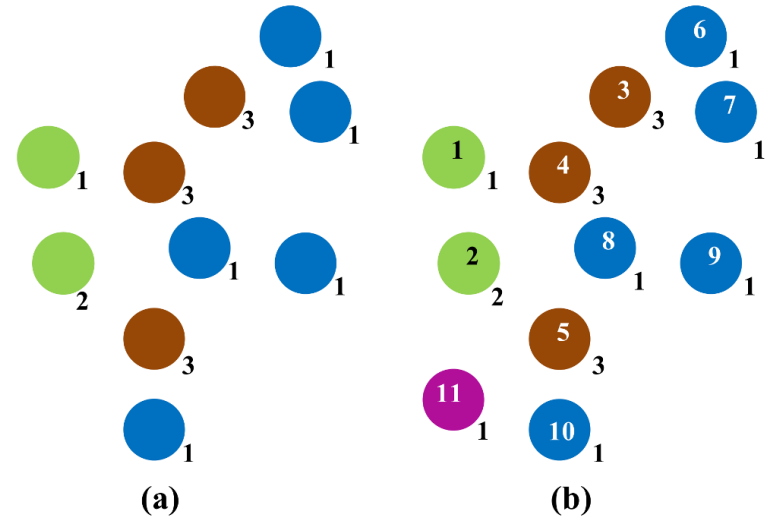
$\sum_k n_k = \sum_k m_k - c \Longrightarrow n = m - c \Longrightarrow c = m - n,$

es decir, tenemos $(m - n)$ componentes conexas, que son árboles porque no tienen ciclos, como queríamos probar. ■

Nótese que esta representación produce varias permutaciones que codifican el mismo individuo. Por ejemplo, la permutación (6,1,**3**,4,5,8,7,9,**2**) representa el mismo individuo que la permutación (6,1,**2**,4,5,8,7,9,**3**) (Figura. 3.3(e)), ya que tanto

$padres_3$ y $padres_2$ representan el nodo número 2. Estas posiciones se denominan posiciones redundantes y eliminamos la redundancia para evitar un aumento innecesario del tamaño del espacio de búsqueda. Para ello, siempre elegimos el individuo cuyos números en las posiciones redundantes están ordenados de menor a mayor, es decir, (6,1,2,4,5,8,7,9,3) en el ejemplo.

Además, nótese que los ciclos de longitud uno, es decir, los ciclos que indican que un nodo es su propio padre, son muy fáciles de detectar con nuestra representación. Por ejemplo, en lo que respecta al problema de la Figura 3.3, sabemos que los números 4 o 5 no pueden ocupar la primera posición de la permutación, porque esto indicaría que $padres_4$ o $padres_5$, es decir, el nodo 3, es el padre de $hijos_1$, que es también el nodo 3. Para ciclos más largos, hay que recorrer la permutación y construir los árboles que ésta codifica para identificar cualquier ciclo. Para ello, utilizamos el algoritmo rápido de unión ponderada (*weighted quick-union algorithm*) (Sedgewick y Wayne, 2011). El peor caso de orden de crecimiento de todas las operaciones de este algoritmo es $\log n$, donde n es la longitud de la permutación.



posición	1	2	3	4	5	6	7	8	9
padres	**1**	**2**	**2**	**3**	**3**	**4**	**4**	**5**	**5**

(c)

posición	1	2	3	4	5	6	7	8	9
hijos	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**

(d)

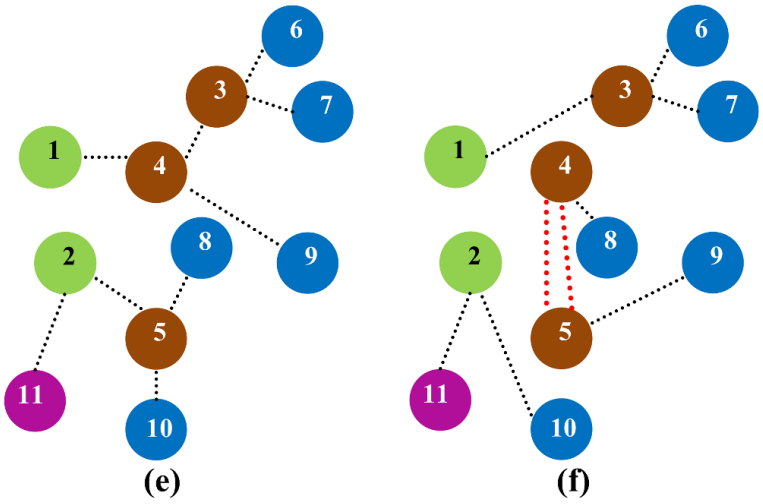


Figura 3.3: Ejemplo de la representación basada en permutaciones. (a) Nodos de un grafo de ejemplo. El grado máximo permitido g_v se especifica a la derecha de cada nodo. El rol de cada nodo está indicado en diferentes colores: los nodos raíz en verde, los nodos intermedios en color marrón y los nodos hoja en azul. (b) Nodos numerados. De acuerdo con la Ecuación (3.2) ($|F| = 3 + 9 - 2 \cdot 3 - 5 = 1$), se necesita un nodo ficticio para resolver el problema, por lo que se agrega el nodo 11 (en rosa). (c)-(d) Matrices auxiliares *padres* e *hijos* necesarias para determinar qué bosque representa cada permutación. (e) Ejemplo de indiviuo correcto, permutación (6,1,2,4,5,8,7,9,3). (f) Ejemplo de individuo inválido ya que contiene un ciclo, permutación (1,8,6,4,5,7,9,2,3).

4

Técnicas de computación evolutiva

La computación evolutiva es una rama de la inteligencia artificial que se inspira en los mecanismos de la evolución biológica para tratar la resolución de problemas de optimización. Los algoritmos evolutivos pueden considerarse técnicas metaheurísticas o estocásticas de optimización global, que se distinguen por el uso de una población de soluciones candidatas a las que se las hace evolucionar mediante procesos iterativos inspirados en los principios de Charles Darwin (Darwin, 1859).

En este trabajo se utilizaron ocho algoritmos de computación evolutiva diferentes para resolver una amplia variedad de problemas sintéticos simulados y posteriormente comparar los resultados obtenidos con los mismos. Los algoritmos analizados fueron: algoritmo de estimación de distribuciones, que denotaremos por EDA (*Estimation of Distribution Algorithm*) (Larrañaga y Lozano, 2002), algoritmo genético generacional, que denotaremos por gGA (*generational Genetic Algorithm*) (Cobb y Grefenstette, 1993), algoritmo genético de estado estacionario, que denotaremos por ssGA (*steady-state Genetic Algorithm*) (Syswerda, 1991), estrategia evolutiva basada en la matriz de covarianzas, que denotaremos por CMAES (*Covariance Matrix Adaptation Evolution Strategy*) (Hansen y Ostermeier, 1996), evolución diferencial, que denotaremos por DE (*Differential Evolution*) (Storn y Price, 1997), estrategia evolutiva elitista y no elitista, que denotaremos por ElitistES y NonElitistES (*Elitist and Non-Elitist Evolution Strategy*) (Rechenberg, 1973) y optimización por enjambre de partículas, que denotaremos PSO (*Particle Swarm Optimization*) (Kennedy y Eberhart, 1995). A continuación se explica cada uno de ellos.

4.1. Algoritmo de estimación de distribuciones (EDA)

Los algoritmos de estimación de distribuciones (EDAs) son métodos de optimización estocásticos que guían la búsqueda del óptimo mediante la construcción y el muestreo de modelos probabilísticos explícitos de soluciones candidatas prometedoras. En los EDAs las interrelaciones entre las variables representando a los individuos se expresan de manera explícita a través del modelo probabilístico asociado con los individuos seleccionados en cada generación. La estimación de dicho modelo es la principal dificultad de esta aproximación.

El pseudo-código de un EDA genérico puede verse en el Algoritmo 1. Partimos de una población inicial de M individuos, pob_0, generada al azar (línea 1). Cada uno de estos individuos es evaluado (línea 2). A continuación y hasta que se verifique una condición de parada (línea 4) se repiten los siguientes pasos: se selecciona un número N ($N \leq M$) de individuos (habitualmente aquellos con mejor aptitud) (línea 6). Como siguiente paso, se estima el modelo probabilístico subyacente a los individuos seleccionados (línea 7). Después, se obtiene la nueva población de tamaño M por medio del muestreo de la distribución de probabilidad aprendida (línea 8). Por último, se evalúa la nueva población.

Algoritmo 1 Pseudo-código de un EDA genérico

1: $\text{pob}_0 \longleftarrow$ GenerarPoblacionInicial(M)
2: Evaluar(pob_0)
3: $t \longleftarrow 0$
4: **mientras** ! CondicionParada() **hacer**
5: $t = t + 1$
6: $\text{pob}_{t-1}^{sel} \longleftarrow$ Seleccionar(N, pob_{t-1})
7: $\text{modelo}_t \longleftarrow$ EstimarModelo(pob_{t-1}^{sel})
8: $\text{pob}_t \longleftarrow$ Muestrear(M, modelo_t)
9: Evaluar(pob_t)
10: **fin mientras**

4.2. Algoritmos genéticos (GAs)

Los algoritmos genéticos son los algoritmos evolutivos más conocidos y más ampliamente utilizados. Inspirados en la evolución biológica, estos algoritmos hacen evolucionar una población de individuos sometiéndola a operaciones aleatorias de

cruce y mutación, así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos que sobreviven (los más adaptados) y cuáles son descartados (los menos aptos).

Tanto EDAs como GAs son algoritmos heurísticos de optimización que basan su búsqueda en el carácter estocástico de la misma y ambos están basados en poblaciones que evolucionan. Sin embargo, mientras que en los EDAs la nueva población de individuos se muestrea de una distribución de probabilidad, en los algoritmos genéticos la evolución de las poblaciones se lleva a cabo por medio de los operadores de cruce y mutación. Mientras que en los EDAs las interrelaciones entre las variables representando a los individuos se expresan de manera explícita, en los algoritmos genéticos dichas interrelaciones se tienen en cuenta de manera implícita.

En este trabajo hemos evaluado el funcionamiento de dos de los GAs más conocidos: el algoritmo genético generacional y el algoritmo genético de estado estacionario, que pasamos a detallar a continuación.

Algoritmo genético generacional (gGA)

El pseudo-código del algoritmo genético generacional (gGA) puede verse en el Algoritmo 2: mientras no se cumpla la condición de parada (línea 3), en cada iteración se seleccionan dos padres de toda la población (línea 5) que se cruzan dando lugar a dos hijos (línea 6) que posteriormente son mutados y evaluados (líneas 7 y 8). Estos individuos recién generados se colocan en una población auxiliar (línea 9) que sustituirá a la población actual cuando esté completamente llena (línea 11), es decir, cuando el número de soluciones generadas sea igual al tamaño de esta población auxiliar. En nuestro caso, los tamaños de la población auxiliar y de la población actual son el mismo. Nótese que esta estrategia podría llevar a perder la mejor solución de la población actual.

Algoritmo genético de estado estacionario (ssGA)

El pseudo-código del algoritmo genético de estado estacionario (ssGA) puede verse en el Algoritmo 3. En cada generación, se seleccionan dos padres de toda la población, con un criterio de selección dado (línea 4). Estos dos individuos se cruzan (línea 5) y después uno de los descendientes obtenidos es mutado (línea 6). El individuo mutado se evalúa y a continuación se inserta en la población, típicamente sustituyendo al peor individuo de la población (si el nuevo es mejor) (línea 8). Este bucle se repite hasta que la condición de parada se cumple (línea 3). Como puede

Algoritmo 2 Pseudo-código de un gGA genérico

```
1: pob ⟵ GenerarPoblacionInicial()
2: Evaluar(pob)
3: mientras ! CondicionParada() hacer
4:    para t = 1 hasta TamañoPob/2 hacer
5:       padres ⟵ Seleccionar(2, pob)
6:       hijos ⟵ Cruzar(probCruce, padres)
7:       hijos ⟵ Mutar(probMutacion, hijos)
8:       Evaluar(hijos)
9:       Añadir(pobAuxiliar, hijos)
10:   fin para
11:   pob ⟵ pobAuxiliar
12: fin mientras
```

observarse, la estrategia ssGA es inherentemente elitista puesto que la mejor solución siempre se conserva.

Algoritmo 3 Pseudo-código de un ssGA genérico

```
1: pob ⟵ GenerarPoblacionInicial()
2: Evaluar(pob)
3: mientras ! CondicionParada() hacer
4:    padres ⟵ Seleccionar(2, pob)
5:    hijo ⟵ Cruzar(probCruce, padres)
6:    hijo ⟵ Mutar(probMutacion, hijo)
7:    Evaluar(hijo)
8:    Reemplazar(pob, hijo)
9: fin mientras
```

4.3. Estrategia evolutiva basada en la matriz de covarianzas (CMAES)

En la estrategia evolutiva basada en la matriz de covarianzas (CMAES), el nombre del algoritmo hace referencia a la estrategia usada para la evolución, ya que las nuevas soluciones candidatas se toman de acuerdo a una distribución normal multivariante que está representada por su matriz de covarianzas. La adaptación de esta matriz es lo que permite aprender al modelo. En esta estrategia apenas se hacen suposiciones sobre la naturaleza de la función objetivo subyacente, siendo aplicable a la mayoría de las situaciones. El inconveniente principal de CMAES es su complicado ajuste de los parámetros.

El estado de CMAES está dado por los parámetros $\boldsymbol{m} \in \mathbb{R}^n$, $\sigma > 0$ y $\boldsymbol{C} \in \mathbb{R}^{nxn}$ de su distribución normal multivariante $\mathcal{N}(\boldsymbol{m}, \sigma^2\boldsymbol{C})$. El pseudo-código de un CMAES genérico puede verse en el Algoritmo 4. Mientras no se cumpla la condición de parada (línea 2), en cada generación se procede del siguiente modo: se muestrean λ hijos a partir de la distribución normal multivariante $\mathcal{N}(\boldsymbol{m}, \sigma^2\boldsymbol{C})$ (línea 3). Los parámetros $\boldsymbol{m}$ y σ se necesitan como entrada al algoritmo y $\boldsymbol{C}$ comienza siendo la matriz identidad (línea 1). A continuación se evalúan los hijos generados (línea 4) y se seleccionan los μ mejores para formar la población actual (línea 5) (μ y λ son otros dos parámetros del algoritmo que necesitan ser establecidos, $\lambda \geq \mu$). A partir de la nueva población de padres, se actualizan los parámetros de la normal multivariante (línea 6).

Algoritmo 4 Pseudo-código de un CMAES genérico

1: $C \longleftarrow I$
2: **mientras** ! CondicionParada() **hacer**
3: hijos $\longleftarrow$ Muestrear(λ, $\mathcal{N}(\boldsymbol{m}, \sigma^2\boldsymbol{C})$)
4: Evaluar(hijos)
5: pob $\longleftarrow$ Seleccionar(μ, hijos)
6: ActualizarParametros(pob, $\boldsymbol{m}$, σ, $\boldsymbol{C}$)
7: **fin mientras**

4.4. Evolución diferencial (DE)

La evolución diferencial (DE) es un algoritmo evolutivo que enfatiza la mutación, utilizando un operador de cruce a posteriori. Esta estrategia fue propuesta para problemas de optimización con parámetros reales. Los individuos de la población son vectores n-dimensionales que representan las soluciones en el espacio de búsqueda. La generación de nuevos individuos se lleva a cabo mediante operadores diferenciales de mutación y cruce.

El pseudo-código de un algoritmo DE genérico puede verse en el Algoritmo 5. Comenzamos con una población de M individuos (línea 1). Mientras no se cumpla la condición de parada, para obtener la población de la siguiente generación se itera del siguiente modo: para cada individuo de la población actual, se seleccionan tres padres de la población (línea 5) y se les aplica mutación diferencial (línea 6). Mediante la mutación diferencial se añade a un individuo elegido aleatoriamente (padres_1), la diferencia proporcional de dos individuos (padres_2 y padres_3) también elegidos aleatoriamente de la población. El nuevo individuo se denomina individuo

mutado o vector mutado. La constante de mutación $\rho > 0$ establece el rango de diferenciación entre los individuos padres_2 y padres_3 con el objetivo de evitar el estancamiento en el proceso de búsqueda.

Tras la mutación, se realiza una operación de cruce sobre cada individuo i de la poblacion (pob_i) para generar un hijo hijos_i. El hijo hijos_i es construido mezclando las componentes del vector mutado y del individuo de la poblacion pob_i, bajo una probabilidad predefinida $\text{C}_r \in [0, 1]$ (líneas 8-14). Finalmente el operador de selección decide en base a la mejor aptitud, si el hijo generado es aceptado y reemplaza al individuo de la poblacion actual pob_i; o si por el contrario, el hijo es rechazado y se conserva el individuo de la población actual en la siguiente generación (línea 18).

Algoritmo 5 Pseudo-código de una DE genérica

```
 1: pob ⟵ GenerarPoblacionInicial(M)
 2: Evaluar(pob)
 3: mientras ! CondicionParada() hacer
 4:    para i = 1 hasta M hacer
 5:       padres ⟵ Seleccionar(3,pob)
 6:       vectorMutado = padres_1 + ρ (padres_2 - padres_3)
 7:       gen ⟵ Aleatorio(1,n)
 8:       para k = 1 hasta n hacer
 9:          si (Aleatorio(0,1) < C_r) OR (k=gen) entonces
10:             hijos_i(k) ⟵ vectorMutado(k)
11:          si no
12:             hijos_i(k) ⟵ pob_i(k)
13:          fin si
14:       fin para
15:    fin para
16:    Evaluar(hijos)
17:    para i = 1 hasta M hacer
18:       pob_i ⟵ Mejor(hijos_i,pob_i)
19:    fin para
20: fin mientras
```

4.5. Estrategias evolutivas (ES)

Las estrategias evolutivas utilizan únicamente operadores de mutación y selección para hacer evolucionar a la población, es decir, no tienen operador de cruce. Como en otros algoritmos evolutivos, los operadores se aplican en un bucle. Cada iteración del bucle es una generación y se continúa hasta que se cumpla un criterio de parada.

La estrategia evolutiva más simple opera sobre una población de tamaño dos: el

padre actual y el resultado de su mutación. Sólo si la aptitud del individuo mutado es al menos tan buena como la del padre, éste se convierte en el padre de la siguiente generación; de lo contrario, el individuo mutado es descartado. Esta estrategia suele denominarse (1+1)-ES. En (1+λ)-ES se generan λ individuos mutados a partir del padre que compiten con él. La estrategia (1,λ)-ES es similar pero en este caso el padre siempre es descartado y el mejor individuo mutado de entre los λ generados a partir de él, es el que se convierte en padre de la próxima generación. Nosotros analizamos aquí las estrategias (μ+λ)-ES y (μ, λ)-ES donde se obtienen λ individuos mutados a partir de una población de μ padres.

Estrategia evolutiva elitista (ElitistES)

La estrategia evolutiva elitista (ElitistES) analizada en este trabajo es (μ+λ)-ES. Esta estrategia se denomina elitista ya que la mejor solución siempre se conserva. El pseudo-código de una ElitistES genérica puede verse en el Algoritmo 6. Comenzamos con una población de tamaño μ (línea 1) y, hasta que no se cumpla cierta condición de parada (línea 3), en cada generación generamos λ individuos mutados (línea 4) a partir de la población actual. Después de la generación de los individuos mutados, tenemos un total de (μ + λ) individuos, que incluyen los padres y los individuos generados a partir de ellos (línea 6). De entre todos, se seleccionan los μ mejores individuos como padres de la próxima generación (línea 7).

Algoritmo 6 Pseudo-código de una ElitistES genérica

```
1: pob ⟵ GenerarPoblacionInicial(μ)
2: Evaluar(pob)
3: mientras ! CondicionParada() hacer
4:    hijos ⟵ Mutar(λ, pob)
5:    Evaluar(hijos)
6:    Añadir(pob, hijos)
7:    pob ⟵ Seleccionar(μ, pob)
8: fin mientras
```

Estrategia evolutiva no elitista (NonElitistES)

La estrategia evolutiva no elitista (NonElitistES) analizada es (μ, λ)-ES. En una NonElitistES los mejores μ individuos mutados de entre los λ generados son seleccionados como padres de la siguiente generación, es decir, ninguno de los μ padres sobrevive a la siguiente generación. En (μ, λ)-ES se tiene que asegurar que $\lambda \geq \mu$. El pseudo-código de una NonElitistES genérica puede verse en el Algoritmo 7.

Algoritmo 7 Pseudo-código de una NonElitistES genérica

```
1: pob ⟵ GenerarPoblacionInicial(μ)
2: Evaluar(pob)
3: mientras ! CondicionParada() hacer
4:    hijos ⟵ Mutar(λ, pob)
5:    Evaluar(hijos)
6:    pob ⟵ Seleccionar(μ, hijos)
7: fin mientras
```

4.6. Optimización por enjambre de partículas (PSO)

La optimización por enjambre de partículas (PSO) es un algoritmo de optimización heurístico que evoca el comportamiento de los enjambres de abejas en la naturaleza (a la hora de buscar alimento, las abejas buscan la región del espacio en la que existe más densidad de flores, porque la probabilidad de que haya polen es mayor). PSO permite optimizar un problema a partir de una población de soluciones candidatas, denominadas partículas, moviendo estas partículas por todo el espacio de búsqueda de acuerdo con fórmulas matemáticas simples que tienen en cuenta la posición de las partículas y su velocidad. El movimiento de cada partícula está influenciado por su mejor posición local conocida, así como por las mejores posiciones globales encontradas por otras partículas a medida que recorren el espacio de búsqueda. Con esto se espera que el enjambre se mueva hacia las mejores soluciones.

Cada partícula se comunica con un entorno que puede ser parte o todo el enjambre y que puede variar dinámicamente. Cada partícula del enjambre lleva asociada su posición, su velocidad, la posición de la mejor solución local encontrada y la mejor posición obtenida por cualquier partícula del entorno.

El pseudo-código de un PSO genérico puede verse en el Algoritmo 8. Se inicializan aleatoriamente todas las partículas (línea 1) (se les asigna una posición y una velocidad aleatorias). Tras cada iteración hay que recalcular la velocidad de cada partícula teniendo en cuenta su mejor posición y la mejor posición de su entorno (línea 4), y actualizar la posición de la partícula a partir de la nueva velocidad (línea 5). A continuación, se actualiza la mejor solución encontrada por la partícula t (siempre que la solución obtenida en la iteración actual sea mejor que la mejor solución local conocida hasta ese momento) (línea 7). Así mismo se actualiza la mejor posición del entorno de la partícula t (línea 8) si la solución obtenida por alguna de las partículas es mejor que la mejor conocida hasta ese momento. Este proceso se repite iterativamente hasta que se cumple cierto criterio de parada (línea 2).

Algoritmo 8 Pseudo-código de un PSO genérico

```
1: particulas ⟵ GenerarPoblacionInicial()
2: mientras ! CondicionParada() hacer
3:    para t = 1 hasta NumParticulas hacer
4:       CalcularVelocidad(particulas_t)
5:       CalcularPosicion(particulas_t)
6:       Evaluar(particulas_t)
7:       ActualizarPosicionLocal(particulas_t)
8:       ActualizarPosicionGlobal(particulas_t)
9:    fin para
10: fin mientras
```

5

Enfoque para la resolución del problema

Para la resolución del problema DRCMST se utilizaron los ocho algoritmos descritos en el Capítulo 4. Los algoritmos genéticos han sido ampliamente estudiados para resolver problemas de optimización basados en permutaciones (Larrañaga et al., 1999) y es conocido que se comportan satisfactoriamente (Bielza et al., 2010; Reeves, 1995; Ruiz y Maroto, 2005). Sin embargo, aunque recientemente se han publicado varios trabajos utilizando modelos probabilísticos con EDAs (Aledo et al., 2013; Ceberio et al., 2011, 2014), EDAs no han sido tan ampliamente desarrollados para problemas de optimización con permutaciones (Ceberio et al., 2012). El modelo probabilístico aprendido en un EDA se espera que refleje la estructura del problema y, por lo tanto, este enfoque debería proporcionar una explotación más eficaz de soluciones prometedoras que los operadores de cruce y mutación de los algoritmos genéticos. Por esta razón, estábamos particularmente interesados en la comparación de EDA, gGA y ssGA.

Por otro lado, aunque CMAES, DE, ElitistES, NonElisitisES y PSO fueron diseñados para resolver problemas basados en valores reales, también quisimos poner a prueba su rendimiento con el problema DRCMST. Para aplicar estos algoritmos se implementó el algoritmo de claves aleatorias (Bean, 1994) por medio del cual podemos codificar un vector de valores reales en una permutación. Una permutación se puede obtener de un vector real $(x_1, x_2, ..., x_n)$ de longitud n ordenando las posiciones utilizando los valores x_i, $i = 1, ..., n$. Por ejemplo, el vector real (5.12, 3.48, 10.21, 1.07, 0.75, 8.54) representaría la permutación (4, 3, 6, 2, 1, 5).

Con el fin de comparar el rendimiento de estos algoritmos, utilizamos el entorno jMetal (Durillo y Nebro, 2011). jMetal es sinónimo de **al**goritmos **Met**aheurísticos

en **j**ava, y es una plataforma orientada a objetos basada en java dirigida al desarrollo, experimentación y estudio de metaheurísticas para la resolución de problemas de optimización uni- y multi-objetivo. Se encuentra bajo la Licencia Pública General de GNU y se puede obtener libremente de `http://jmetal.sourceforge.net`. jMetal proporciona un rico conjunto de clases que pueden ser utilizadas como bloques de construcción de metaheurísticas; haciendo uso de la reutilización de código, los algoritmos comparten los mismos componentes básicos, lo que facilita no sólo el desarrollo de nuevas técnicas, sino también el llevar a cabo diferentes tipos de estudios.

En la Figura 5.1 se utiliza el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modelling Language*) para describir la arquitectura y los componentes de jMetal. La figura muestra un diagrama de clases UML que representa los componentes principales y sus relaciones. El diagrama es una versión simplificada con el fin de hacerlo comprensible. La arquitectura básica de jMetal se basa en un *algoritmo* que resuelve un *problema* haciendo uso de uno o más conjuntos de soluciones (*SolutionSet*) y un conjunto de objetos *operador*. jMetal utiliza una terminología genérica para nombrar a las clases con el fin de hacerlas útiles para cualquier metaheurística. En el contexto de los algoritmos evolutivos, las poblaciones y los individuos corresponden a los objetos *SolutionSet* y *Solution* de jMetal, respectivamente.

Encontramos que jMetal contenía todos los algoritmos en los que estábamos interesados, salvo EDA. Por lo tanto, añadimos nuestra propia implementación de esta técnica al entorno (ver detalles en la Sección 5.1). Teniendo en cuenta que la posición absoluta de cada número en la permutación está estrechamente relacionado con el rendimiento en nuestro problema (ya que cada número en la permutación denota un eje seleccionado para construir el bosque), implementamos el algoritmo *node histogram sampling* presentado en Tsutsui (2006). Este algoritmo modela las frecuencias de los números en cada posición absoluta en los individuos de una población.

Tanto para el EDA implementado como para los otros algoritmos uni-objetivo ya incluidos en jMetal, hicimos mejoras debido a las características específicas de nuestra representación. Por un lado, descartamos la generación de individuos que contienen ciclos de longitud uno (que son fáciles de detectar como se describe en el Capítulo 3) y, por otro lado, eliminamos la redundancia de nuestra representación seleccionando un individuo representativo entre los individuos redundantes, como se explicó anteriormente.

Se utilizaron los parámetros por defecto proporcionados en jMetal para todos los algoritmos. Para cada problema, se estableció un tamaño de población igual a $10|V|$

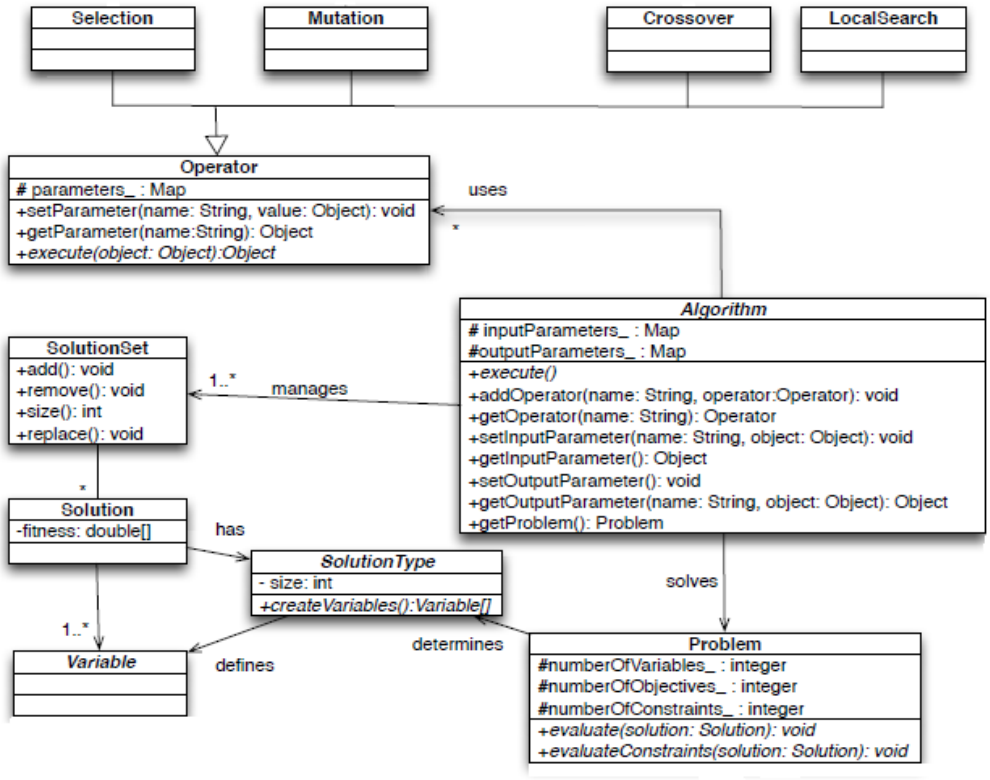


Figura 5.1: jMetal. Diagrama de clases. Diagrama de clases UML de jMetal que representa los componentes principales y sus relaciones.

para todos los algoritmos. Para cada ejecución de cada problema, la población inicial se generó aleatoriamente incluyendo las mejoras mencionadas anteriormente (ciclos de longitud uno y redundancia). Decidimos parar la ejecución de los algoritmos si no había mejora en la mejor aptitud de más de un 0.1 % en los últimas 500 generaciones.

5.1. Implementación en jMetal

Se desarrolló y se incorporó al entorno jMetal todo el código necesario para resolver un problema DRCMST. Partiendo de un problema definido como en la Ecuación (2.1) se generan las matrices auxiliares *padres* e *hijos* y se comprueba si es necesario añadir nodos ficticios para utilizar la representación propuesta. La evaluación de cada individuo de la población se realiza recorriendo la permutación que representa dicho individuo formando el bosque codificado (cada posición de la permutación es un eje en el bosque). La estructura de datos utilizada para almacenar los bosques que se van formando al analizar las permutaciones se extrajo de Sedgewick y Wayne (2011). Se implementó también la detección de ciclos de longitud uno y la

eliminación de la redundancia en la población.

Como se ha comentado, las metaheurísticas en jMetal se ejecutan haciendo uso de un conjunto de objetos *operador* (selección, mutación, cruce...). Nosotros añadimos dos nuevos operadores que fueron utilizados en todos los algoritmos evolutivos analizados. Por un lado, se implementó el operador *Generator* con el que creamos una población inicial aleatoria, al igual que se hacía previamente en el entorno jMetal, pero añadiendo la restricción de que los individuos generados no tengan ciclos de longitud uno. Se sabe que estos individuos son inválidos por lo que, de este modo, evitamos algunos de los individuos no válidos que pueden aparecer en la población inicial. Por otro lado, se implementó el operador *StopCondition*. En jMetal todas las metaheurísticas terminaban su ejecución cuando se llegaba a un número máximo de iteraciones. Con el operador *StopCondition* damos la opción de elegir si queremos que la condición de parada sea un número máximo de iteraciones o si queremos parar cuando no se produzca una mejoría relativa mínima en un número dado de iteraciones del algoritmo. Nosotros nos decantamos por la segunda opción.

Se ampliaron las metaheurísticas uni-objetivo del entorno jMetal incluyendo la implementación de un EDA genérico. En resumen, se genera la población inicial y, mientras no se cumpla la condición de parada, se seleccionan los individuos de la población de acuerdo a su aptitud; a continuación, se aprende la distribución de probabilidad a partir de los individuos seleccionados; se generan nuevos individuos mediante el muestreo de la distribución de probabilidad aprendida y se genera una nueva población siguiendo cierto criterio de reemplazo.

Para la implementación del EDA fue necesario crear los operadores *Learning*, *Sampling* y *Replacement*. En concreto, en el EDA implementado (*node histogram sampling*), el operador de aprendizaje de la distribución de probabilidad genera una matriz con las frecuencias de los números en cada posición absoluta en los individuos de la población seleccionada. El operador de muestreo genera una población de individuos a partir de la matriz aprendida en el paso anterior. Como mejora añadida debida a las características específicas de nuestra representación, añadimos los individuos generados siempre que no tengan ciclos de longitud uno y les eliminamos la redundancia. Esta mejora se incluyó en todas las metaheurísticas analizadas. Por ejemplo, en los algoritmos genéticos, si una operación de cruce o mutación produce un individuo con ciclos de longitud uno, éste es descartado. Además, se elimina la redundancia de todo individuo válido generado. En cuanto al operador de reemplazo, implementamos un reemplazo elitista, es decir, creamos una nueva población uniendo los individuos seleccionados por el operador de selección con los mejores individuos generados por el operador de muestreo. Este método de reemplazo es

apropiado para la selección por truncamiento. Puesto que este tipo de selección no estaba incluida en los operadores de selección de jMetal, la incorporamos nosotros. En la selección por truncamiento, las soluciones se ordenan según su aptitud y se selecciona una proporción (50 % por defecto en nuestro caso) de los individuos más aptos.

5. ENFOQUE PARA LA RESOLUCIÓN DEL PROBLEMA

6

Generación de problemas de prueba

Simulamos diez problemas DRCMST para cada uno de los siguientes tamaños $|V|$= 20, 40, 60, 80, 100, 200 nodos. El número de raíces y nodos intermedios se generó al azar, aunque se incluyeron algunas restricciones. El número de nodos raíz de un problema se limitó al 20 % del total de nodos del problema, es decir, $|R| \leq 0{,}2|V|$. El número de nodos intermedios se limitó al 75 % del total de nodos del problema ($|I| \leq 0{,}75|V|$). El número de nodos hoja se obtuvo como $|H| = |V| - |R| - |I|$. El grado máximo permitido para las raíces y los nodos intermedios (los nodos hoja siempre tienen un grado igual a 1) también se simuló al azar de la siguiente manera para cada nodo: el grado máximo permitido g_v se fijó entre 1 y 4 para los nodos raíz y entre 2 y 5 para los nodos intermedios.

Simulamos dos tipos de problemas: cinco problemas de cada tamaño con distancias euclídeas (Tabla 6.1) y cinco problemas de cada tamaño con distancias obtenidas aleatoriamente (Tabla 6.2). El conjunto de problemas con distancias euclídeas se creó mediante la simulación de las coordenadas (x, y, z) de cada punto entre $x_{min} = y_{min} = z_{min} = 1$ y $x_{max} = y_{max} = z_{max} = 100$. Después, se obtuvo la matriz de costes con las distancias euclídeas reales entre pares de puntos. Para todos los problemas con distancias aleatorias, se obtuvo la matriz de costes mediante la simulación de números aleatorios entre 1 y 100 entre cada par de puntos. Al evaluar los individuos (permutaciones) de nuestra población con ambos tipos de matrices de costes, fue preferible un valor de aptitud bajo.

Las Tablas 6.1 y 6.2 muestran las características de cada uno de los problemas simulados. Para cada problema, se muestra el número de nodos de cada rol y la longitud de la permutación que representa el bosque. Nótese que la longitud de la

Tabla 6.1: Descripción de los problemas simulados con distancias euclídeas. La tabla muestra el número de raíces, nodos intermedios, nodos hoja y nodos ficticios y la longitud de las permutaciones que codifican las soluciones de los problemas.

Distancias euclídeas	$\|R\|$ raíces	$\|I\|$ intermedios	$\|H\|$ hojas	$\|F\|$ ficticios	n long. permutación
Problema 1-20Nodos	3	9	8	10	27
Problema 2-20Nodos	3	6	11	0	17
Problema 3-20Nodos	1	9	10	3	22
Problema 4-20Nodos	3	11	6	10	27
Problema 5-20Nodos	2	12	6	11	29
Problema 1-40Nodos	5	18	17	15	50
Problema 2-40Nodos	7	12	21	6	39
Problema 3-40Nodos	4	22	14	16	52
Problema 4-40Nodos	4	18	18	8	44
Problema 5-40Nodos	7	16	17	13	46
Problema 1-60Nodos	5	24	31	1	56
Problema 2-60Nodos	1	36	23	13	72
Problema 3-60Nodos	11	17	32	8	57
Problema 4-60Nodos	9	26	25	18	69
Problema 5-60Nodos	11	23	26	22	71
Problema 1-80Nodos	13	42	25	50	117
Problema 2-80Nodos	11	25	44	1	70
Problema 3-80Nodos	13	36	31	36	103
Problema 4-80Nodos	12	33	35	5	73
Problema 5-80Nodos	15	29	36	21	86
Problema 1-100Nodos	15	45	40	29	114
Problema 2-100Nodos	7	56	37	39	132
Problema 3-100Nodos	6	52	42	14	108
Problema 4-100Nodos	18	42	40	39	121
Problema 5-100Nodos	15	56	29	49	134
Problema 1-200Nodos	23	74	103	24	201
Problema 2-200Nodos	30	100	70	91	261
Problema 3-200Nodos	11	102	87	43	232
Problema 4-200Nodos	10	103	87	16	206
Problema 5-200Nodos	20	99	81	59	239

permutación para cada problema depende del número de nodos de cada rol y de su grado máximo permitido.

Se obtuvo una amplia variedad de problemas. El número de árboles en el bosque varió de un solo árbol ($|R| = 1$, seis veces de 60) a 30 árboles (problema número 2 con 200 nodos y distancias euclídeas). El número de nodos intermedios osciló entre el 25 % y el 70 % de todos los nodos de la red. El número de nodos hoja varió entre el 26 % y el 60 % de $|V|$, dependiendo del problema. En uno de los problemas (problema número 2 con 20 nodos y distancias euclídeas) no tuvo que ser añadido ningún nodo

Tabla 6.2: Descripción de los problemas simulados con distancias aleatorias. La tabla muestra el número de raíces, nodos intermedios, nodos hoja y nodos ficticios y la longitud de las permutaciones que codifican las soluciones de los problemas.

Distancias aleatorias	$\|R\|$ raíces	$\|I\|$ intermedios	$\|H\|$ hojas	$\|F\|$ ficticios	n long. permutación
Problema 1-20Nodos	2	11	7	5	23
Problema 2-20Nodos	3	10	7	9	26
Problema 3-20Nodos	3	9	8	3	20
Problema 4-20Nodos	2	9	9	5	23
Problema 5-20Nodos	1	13	6	5	24
Problema 1-40Nodos	7	12	21	3	36
Problema 2-40Nodos	7	22	11	31	64
Problema 3-40Nodos	4	12	24	1	37
Problema 4-40Nodos	4	24	12	23	59
Problema 5-40Nodos	7	10	23	3	36
Problema 1-60Nodos	1	32	27	6	65
Problema 2-60Nodos	2	42	16	30	88
Problema 3-60Nodos	1	35	24	18	77
Problema 4-60Nodos	3	41	16	34	91
Problema 5-60Nodos	11	24	25	30	79
Problema 1-80Nodos	6	43	31	31	105
Problema 2-80Nodos	5	47	28	33	108
Problema 3-80Nodos	14	32	34	24	90
Problema 4-80Nodos	2	55	23	34	112
Problema 5-80Nodos	7	44	29	22	95
Problema 1-100Nodos	11	51	38	28	117
Problema 2-100Nodos	18	42	40	47	129
Problema 3-100Nodos	7	59	34	39	132
Problema 4-100Nodos	10	63	27	58	148
Problema 5-100Nodos	12	38	50	12	100
Problema 1-200Nodos	22	98	80	62	240
Problema 2-200Nodos	16	102	82	50	234
Problema 3-200Nodos	19	99	82	51	232
Problema 4-200Nodos	1	110	89	29	228
Problema 5-200Nodos	4	122	74	60	256

ficticio, mientras que en el problema número 2 con 40 nodos y distancias aleatorias se tuvieron que añadir 31 nodos ficticios (77,5 % del tamaño del problema). En promedio, la longitud de las permutaciones de los problemas con distancias euclídeas fue un 15,6 % mayor que el número de nodos del problema. Para problemas con distancias aleatorias, este promedio se elevó a un 22,8 %. Este aumento es debido exclusivamente al hecho de que los problemas se generaron aleatoriamente.

7

Resultados

Utilizamos los ocho algoritmos descritos en el Capítulo 4 para resolver los 60 problemas simulados descritos en el Capítulo 6. Cada problema se ejecutó 20 veces con cada algoritmo (con una nueva población inicial generada aleatoriamente en cada ejecución). Todos los resultados se obtuvieron utilizando el supercomputador Magerit. Magerit se ubica en el Centro de Supercomputación y Visualización de Madrid (CeSViMa) y consta de un clúster de propósito general con arquitectura dual (Intel y POWER). Nosotros hicimos uso de nodos POWER7 de 3,3 GHz (422,4 GFlops) con 32 GB de RAM y 300 GB de disco duro local.

Las Tablas 7.1 y 7.2 muestran la media y la desviación estándar del valor de aptitud de las mejores soluciones encontradas para cada algoritmo. El mejor valor medio para cada problema se resalta en gris. Tanto para distancias euclídeas como para distancias aleatorias, EDA y los algoritmos genéticos (gGA and ssGA) obtuvieron el menor (mejor) valor de aptitud en todos los casos. EDA es un claro ganador hasta problemas de tamaño 100; sin embargo, gGA logra los mejores resultados para los problemas de tamaño 200. Del mismo modo, las Tablas 7.3 y 7.4 muestran el tiempo de ejecución medio y la desviación estándar (en minutos) para cada algoritmo. ssGA fue el más rápido en los problemas de tamaño 20 y 40 y en la mayoría de los problemas de tamaño 60. Para los problemas más grandes, PSO y DE consiguieron tiempos de ejecución más bajos. Nótese, sin embargo, que, aunque fueron rápidos, estos dos algoritmos no consiguieron buenas soluciones para los problemas analizados.

Las Figuras 7.1, 7.2, 7.3 y 7.4 muestran una comparación de la mejor aptitud media encontrada y el tiempo medio de ejecución entre los algoritmos en los problemas simulados. Para los tiempos de ejecución se decidió no incluir la estrategia EDA porque para problemas grandes necesitó un tiempo de ejecución mucho mayor que

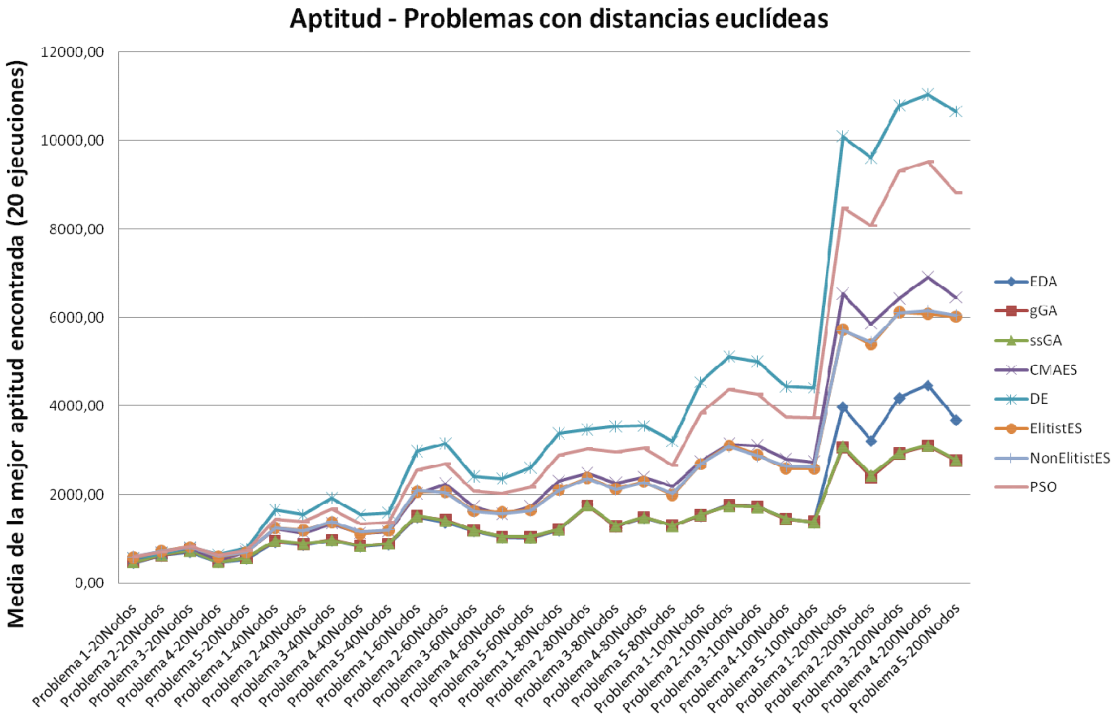


Figura 7.1: Media de la mejor aptitud encontrada por cada uno de los ocho algoritmos evolutivos en el estudio de los 30 problemas analizados con distancias euclídeas. Se realizaron veinte ejecuciones de cada problema para cada algoritmo.

el resto de algoritmos, e incluyéndolo no era posible apreciar las diferencias entre el resto de los algoritmos analizados.

La Figura 7.5(a) muestra los mejores valores de aptitud encontrados por el EDA implementado en las primeras 20 generaciones de una ejecución del problema número 1 con 20 nodos y distancias euclídeas. Para este problema, estábamos interesados en la construcción de un bosque de tres árboles. Las Figuras 7.5(b)-(f) muestran los árboles que representan los mejores individuos encontrados en las generaciones 1, 5, 10, 15 y 20. Una vez más, las raíces se representan en verde, los nodos intermedios se muestran en color marrón y los nodos hoja en azul. En cada bosque, los ejes que difieren del mejor bosque encontrado por el algoritmo se muestran en rojo. Observamos que el número de ejes de color rojo desaparece gradualmente a medida que el número de generaciones aumenta porque el algoritmo se va acercando a la mejor solución encontrada. El bosque obtenido en la generación 20 no tiene ningún eje rojo porque el algoritmo no mejoró después de esta generación, es decir, en la generación 20 se obtuvo el mejor valor de aptitud para este problema.

Se aplicó el test de Friedman para detectar diferencias estadísticamente sig-

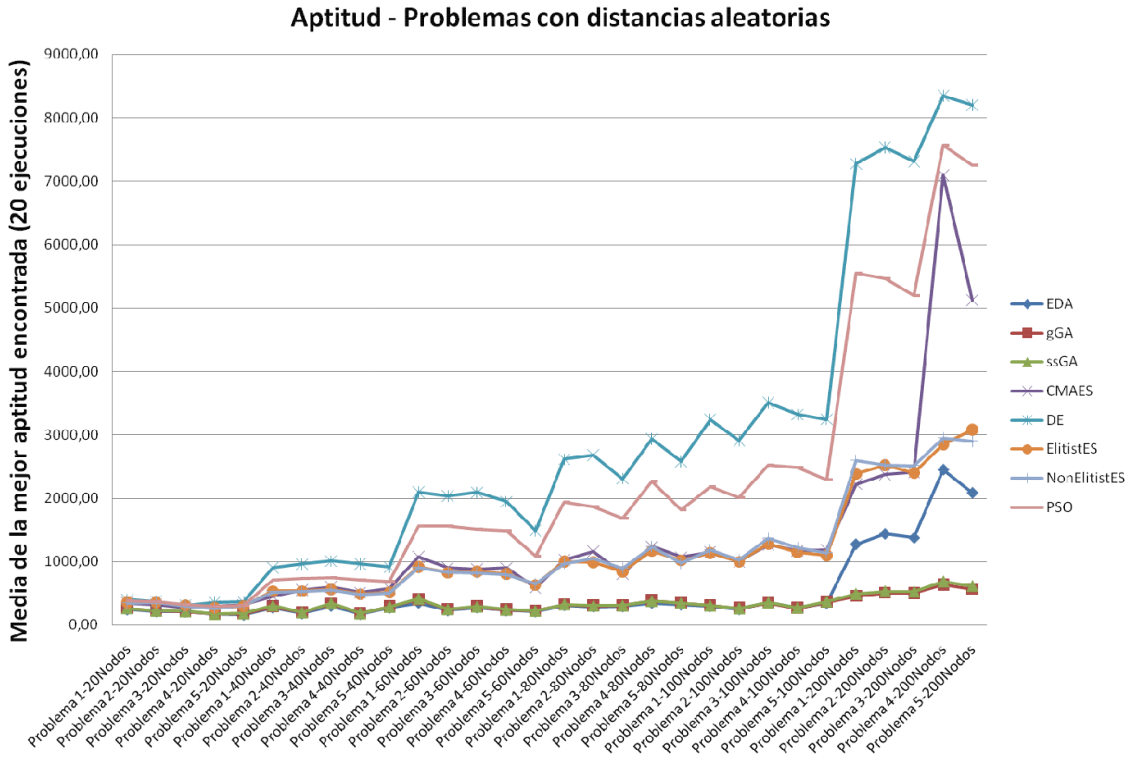


Figura 7.2: Media de la mejor aptitud encontrada por cada uno de los ocho algoritmos evolutivos en el estudio de los 30 problemas analizados con distancias aleatorias. Se realizaron veinte ejecuciones de cada problema para cada algoritmo.

nificativas teniendo en cuenta el conjunto global de algoritmos (Friedman, 1937). Se utilizó el paquete MULTIPLETEST disponible en la página web SCI2S `http://sci2s.ugr.es/sicidm/`. Se aplicó el test de Friedman cuatro veces: una vez para la mejor aptitud media encontrada por los ocho algoritmos en los 30 problemas con distancias euclídeas (Tabla 7.1), otra vez para el tiempo de ejecución medio de los ocho algoritmos en los 30 problemas con distancias euclídeas (Tabla 7.3), y otras dos veces en las mismas instancias para el grupo de problemas con distancias aleatorias (Tablas 7.2 y 7.4). El test de Friedman rechazó la hipótesis nula de igualdad, para la aptitud y para el tiempo de ejecución, tanto para distancias euclídeas como para distancias aleatorias (p-valor $\leq 10^{-10}$ en todos los casos). Una vez que la hipótesis nula de igualdad entre todos los pares de algoritmos fue rechazada, se aplicó el procedimiento de Bergmann-Hommel para comparaciones múltiples (Bergmann y Hommel, 1988) para encontrar cuales eran las parejas en concreto donde se producían las diferencias.

La Tabla 7.5 muestra los p-valores del procedimiento de Bergmann-Hommel para

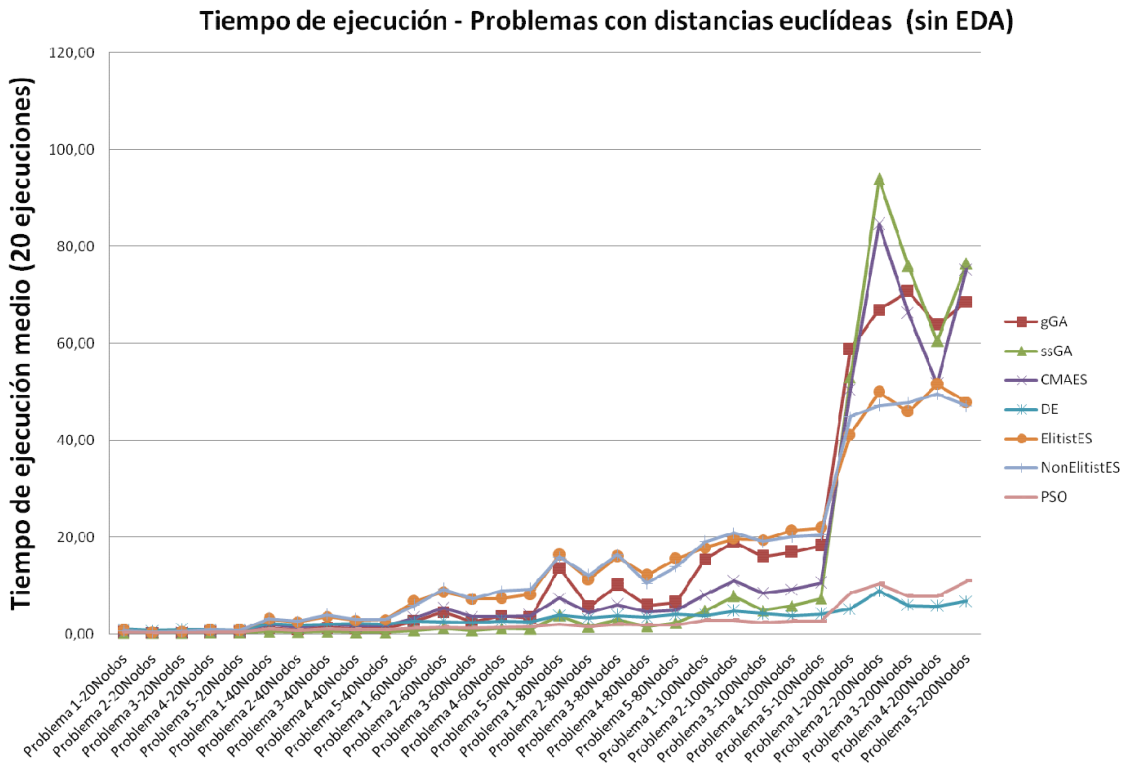


Figura 7.3: Media del tiempo de ejecución (en minutos) para los 30 problemas analizados con distancias euclídeas. Se realizaron veinte ejecuciones de cada problema para cada algoritmo. EDA no se incluye porque, debido a su alto tiempo de ejecución, no sería posible apreciar las diferencias entre los otros algoritmos.

todas las comparaciones por pares para la aptitud y el tiempo de ejecución tanto para distancias euclídeas como para distancias aleatorias. Los p-valores que no rechazan la igualdad entre un par de algoritmos (p-valores $> 0{,}05$) se muestran en negrita. La Figura 7.6 ilustra los resultados de la comparación múltiple de algoritmos. Estos diagramas fueron introducidos en Demšar (2006) y de una forma clara ilustran las diferencias estadísticamente significativas entre los algoritmos. El test de Friedman clasifica los algoritmos tales que el algoritmo con mejor desempeño tiene rango 1, el segundo mejor rango 2, etc. En los diagramas los rangos más bajos (los mejores) están a la derecha por lo que los algoritmos en el lado derecho pueden considerarse mejores. Los grupos de algoritmos que no son significativamente diferentes están conectados. Analizando las comparaciones por pares, los resultados mostraron que no hubo diferencias significativas en la mejor aptitud entre EDA, gGA y ssGA (filas 1 y 2 en las columnas 1 y 2 de la Tabla 7.5 y diagramas en la parte superior de la Figura 7.6). En cuanto a los tiempos de ejecución, sin embargo, encontramos diferencias significativas entre estos tres algoritmos. EDA necesitó un tiempo de ejecución muy

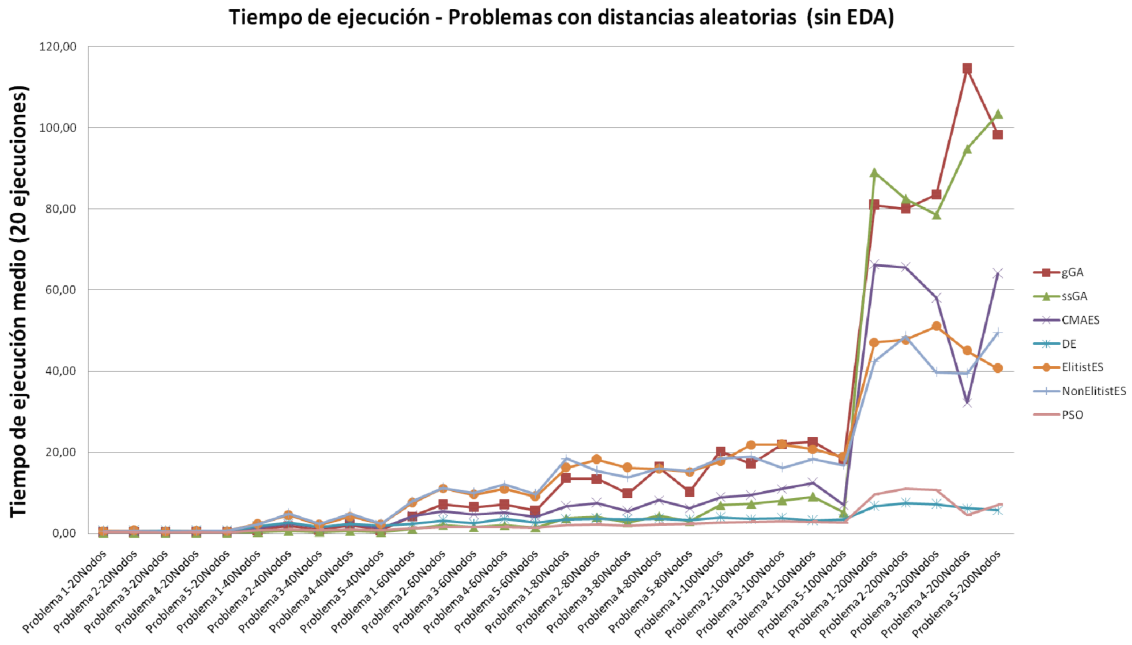


Figura 7.4: Media del tiempo de ejecución (en minutos) para los 30 problemas analizados con distancias aleatorias. Los mismos comentarios que en la Figura 7.3 se aplican aquí.

superior a los dos algoritmos genéticos para grandes problemas (Tablas 7.3 y 7.4). gGA y ssGA tuvieron tiempos de ejecución similares, pero la hipótesis de igualdad fue rechazada (fila 8 en las columnas 3 y 4 de la Tabla 7.5 y diagramas en la parte inferior de la Figura 7.6). Por lo tanto, podríamos concluir que ssGA es preferible porque necesitó un tiempo de ejecución menor.

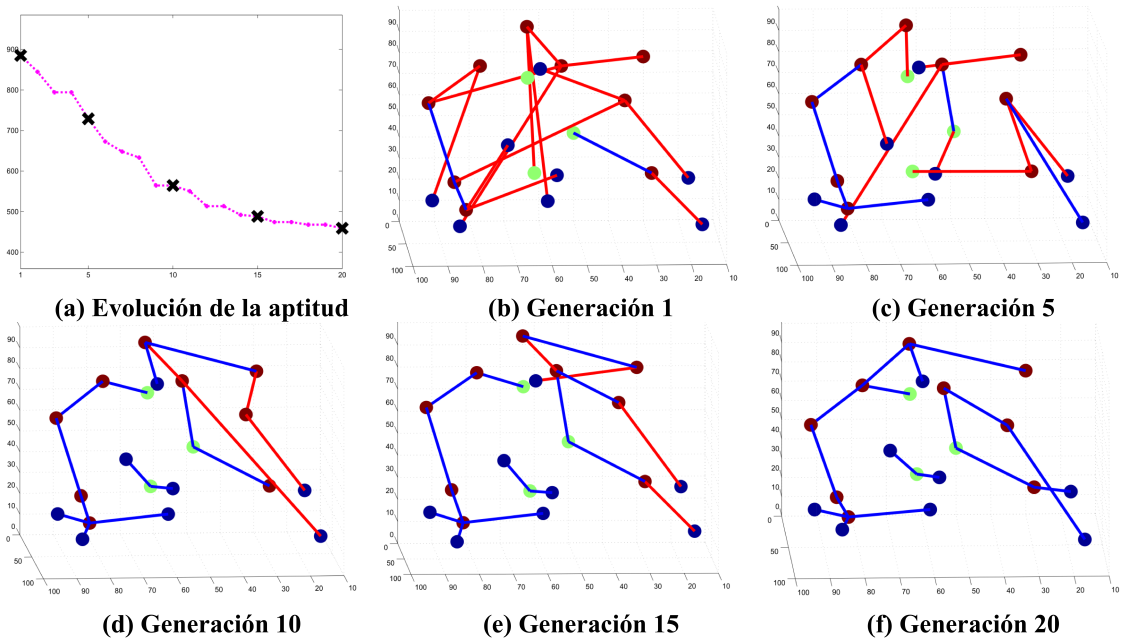


Figura 7.5: Evolución de la mejor aptitud encontrada por EDA en 20 generaciones del problema número 1 con 20 nodos y distancias euclídeas. (a) Evolución de la aptitud en 20 generaciones (las cruces indican la aptitud de los individuos que se muestran en (b)-(f)). (b)-(f) Bosque codificado por las mejores soluciones encontradas en las generaciones 1, 5, 10, 15 y 20. Los nodos raíz se muestran en verde, los nodos intermedios en marrón y los nodos hoja en azul. Los ejes que difieren del mejor bosque encontrado por el algoritmo se muestran en rojo. El algoritmo no mejoró después de la generación 20.

Tabla 7.1: Media y desviación estándar del valor de aptitud ($\bar{x}_{\pm s}$) de las mejores soluciones encontradas por cada algoritmo en 20 ejecuciones de cada problema con distancias euclídeas. El menor valor medio (el mejor) para cada problema está resaltado en gris.

Distancias euclídeas	EDA	gGA	ssGA	CMAES	DE	ElitistES	NonElitistES	PSO
Problema 1-20Nodos	$460{,}99_{\pm 5{,}19}$	$468{,}57_{\pm 11{,}97}$	$471{,}16_{\pm 11{,}13}$	$549{,}13_{\pm 30{,}15}$	$561{,}97_{\pm 25{,}81}$	$573{,}37_{\pm 31{,}18}$	$591{,}21_{\pm 31{,}87}$	$591{,}66_{\pm 56{,}79}$
Problema 2-20Nodos	$629{,}62_{\pm 0{,}00}$	$632{,}07_{\pm 3{,}94}$	$642{,}27_{\pm 9{,}91}$	$683{,}73_{\pm 16{,}72}$	$671{,}22_{\pm 15{,}06}$	$715{,}53_{\pm 34{,}99}$	$707{,}32_{\pm 34{,}23}$	$697{,}62_{\pm 42{,}59}$
Problema 3-20Nodos	$714{,}11_{\pm 6{,}60}$	$732{,}77_{\pm 21{,}35}$	$732{,}86_{\pm 23{,}42}$	$795{,}62_{\pm 27{,}75}$	$809{,}85_{\pm 17{,}96}$	$817{,}81_{\pm 41{,}46}$	$799{,}10_{\pm 30{,}96}$	$828{,}34_{\pm 42{,}77}$
Problema 4-20Nodos	$475{,}08_{\pm 2{,}18}$	$478{,}81_{\pm 5{,}78}$	$477{,}87_{\pm 6{,}00}$	$532{,}09_{\pm 38{,}85}$	$648{,}40_{\pm 34{,}10}$	$596{,}29_{\pm 51{,}93}$	$615{,}87_{\pm 43{,}32}$	$633{,}07_{\pm 48{,}70}$
Problema 5-20Nodos	$552{,}85_{\pm 5{,}26}$	$567{,}49_{\pm 13{,}90}$	$570{,}07_{\pm 11{,}13}$	$702{,}58_{\pm 41{,}25}$	$782{,}91_{\pm 22{,}33}$	$695{,}04_{\pm 48{,}69}$	$703{,}63_{\pm 45{,}28}$	$734{,}19_{\pm 44{,}31}$
Problema 1-40Nodos	$928{,}90_{\pm 9{,}51}$	$945{,}45_{\pm 15{,}09}$	$956{,}12_{\pm 20{,}09}$	$1237{,}15_{\pm 39{,}35}$	$1660{,}12_{\pm 35{,}40}$	$1245{,}14_{\pm 62{,}10}$	$1245{,}56_{\pm 68{,}77}$	$1441{,}12_{\pm 81{,}17}$
Problema 2-40Nodos	$870{,}71_{\pm 7{,}71}$	$881{,}01_{\pm 22{,}92}$	$886{,}97_{\pm 23{,}39}$	$1125{,}46_{\pm 44{,}33}$	$1548{,}21_{\pm 52{,}17}$	$1189{,}10_{\pm 62{,}80}$	$1168{,}29_{\pm 51{,}20}$	$1388{,}96_{\pm 75{,}28}$
Problema 3-40Nodos	$963{,}28_{\pm 14{,}09}$	$972{,}47_{\pm 20{,}65}$	$972{,}91_{\pm 19{,}10}$	$1352{,}74_{\pm 52{,}87}$	$1924{,}88_{\pm 50{,}30}$	$1370{,}71_{\pm 101{,}22}$	$1389{,}86_{\pm 81{,}18}$	$1675{,}16_{\pm 102{,}35}$
Problema 4-40Nodos	$827{,}47_{\pm 10{,}70}$	$839{,}79_{\pm 14{,}35}$	$844{,}35_{\pm 18{,}09}$	$1125{,}82_{\pm 49{,}15}$	$1543{,}02_{\pm 30{,}83}$	$1110{,}94_{\pm 50{,}37}$	$1157{,}69_{\pm 72{,}67}$	$1326{,}22_{\pm 59{,}47}$
Problema 5-40Nodos	$877{,}84_{\pm 10{,}68}$	$884{,}41_{\pm 13{,}69}$	$898{,}27_{\pm 17{,}14}$	$1164{,}23_{\pm 43{,}97}$	$1594{,}48_{\pm 30{,}11}$	$1186{,}15_{\pm 67{,}56}$	$1197{,}98_{\pm 66{,}26}$	$1363{,}30_{\pm 92{,}04}$
Problema 1-60Nodos	$1497{,}78_{\pm 22{,}01}$	$1510{,}12_{\pm 37{,}42}$	$1520{,}22_{\pm 23{,}70}$	$2023{,}22_{\pm 65{,}46}$	$2984{,}57_{\pm 53{,}69}$	$2057{,}55_{\pm 77{,}75}$	$2092{,}61_{\pm 100{,}78}$	$2553{,}75_{\pm 135{,}06}$
Problema 2-60Nodos	$1371{,}37_{\pm 36{,}04}$	$1420{,}77_{\pm 44{,}49}$	$1411{,}11_{\pm 39{,}48}$	$2247{,}22_{\pm 200{,}09}$	$3155{,}52_{\pm 48{,}89}$	$2057{,}06_{\pm 100{,}28}$	$2047{,}64_{\pm 93{,}69}$	$2701{,}65_{\pm 262{,}46}$
Problema 3-60Nodos	$1187{,}42_{\pm 8{,}96}$	$1188{,}15_{\pm 7{,}09}$	$1194{,}80_{\pm 13{,}54}$	$1738{,}09_{\pm 76{,}00}$	$2417{,}27_{\pm 46{,}09}$	$1624{,}98_{\pm 67{,}19}$	$1621{,}39_{\pm 68{,}12}$	$2079{,}24_{\pm 104{,}39}$
Problema 4-60Nodos	$1048{,}51_{\pm 10{,}31}$	$1046{,}42_{\pm 11{,}53}$	$1045{,}92_{\pm 17{,}11}$	$1558{,}62_{\pm 58{,}85}$	$2366{,}25_{\pm 43{,}86}$	$1597{,}14_{\pm 93{,}73}$	$1567{,}48_{\pm 87{,}83}$	$2017{,}42_{\pm 87{,}58}$
Problema 5-60Nodos	$1031{,}98_{\pm 10{,}37}$	$1042{,}64_{\pm 15{,}62}$	$1044{,}00_{\pm 18{,}29}$	$1741{,}26_{\pm 48{,}03}$	$2612{,}38_{\pm 40{,}42}$	$1644{,}51_{\pm 98{,}08}$	$1641{,}77_{\pm 85{,}77}$	$2159{,}81_{\pm 127{,}70}$
Problema 1-80Nodos	$1199{,}72_{\pm 12{,}28}$	$1211{,}43_{\pm 19{,}32}$	$1214{,}93_{\pm 19{,}03}$	$2309{,}58_{\pm 53{,}53}$	$3384{,}96_{\pm 56{,}03}$	$2101{,}21_{\pm 75{,}24}$	$2099{,}97_{\pm 96{,}66}$	$2885{,}96_{\pm 135{,}23}$
Problema 2-80Nodos	$1765{,}59_{\pm 22{,}67}$	$1748{,}81_{\pm 18{,}21}$	$1764{,}12_{\pm 18{,}60}$	$2491{,}55_{\pm 76{,}46}$	$3479{,}38_{\pm 56{,}74}$	$2371{,}97_{\pm 95{,}69}$	$2367{,}21_{\pm 86{,}80}$	$3031{,}89_{\pm 133{,}37}$
Problema 3-80Nodos	$1283{,}39_{\pm 17{,}16}$	$1283{,}89_{\pm 18{,}07}$	$1293{,}63_{\pm 27{,}91}$	$2268{,}18_{\pm 69{,}89}$	$3550{,}09_{\pm 67{,}01}$	$2124{,}02_{\pm 86{,}17}$	$2115{,}55_{\pm 116{,}89}$	$2956{,}60_{\pm 179{,}78}$
Problema 4-80Nodos	$1481{,}12_{\pm 25{,}11}$	$1485{,}13_{\pm 30{,}77}$	$1489{,}83_{\pm 34{,}92}$	$2399{,}71_{\pm 71{,}41}$	$3559{,}76_{\pm 55{,}53}$	$2288{,}49_{\pm 89{,}17}$	$2280{,}17_{\pm 86{,}28}$	$3043{,}38_{\pm 134{,}49}$
Problema 5-80Nodos	$1306{,}43_{\pm 15{,}53}$	$1298{,}42_{\pm 15{,}92}$	$1292{,}81_{\pm 15{,}48}$	$2184{,}25_{\pm 67{,}31}$	$3204{,}32_{\pm 49{,}01}$	$1985{,}62_{\pm 101{,}09}$	$2030{,}74_{\pm 126{,}11}$	$2657{,}66_{\pm 115{,}82}$
Problema 1-100Nodos	$1525{,}71_{\pm 24{,}97}$	$1533{,}76_{\pm 23{,}52}$	$1548{,}26_{\pm 26{,}53}$	$2748{,}73_{\pm 86{,}84}$	$4544{,}48_{\pm 51{,}07}$	$2685{,}81_{\pm 127{,}81}$	$2695{,}13_{\pm 80{,}30}$	$3852{,}29_{\pm 158{,}94}$
Problema 2-100Nodos	$1774{,}14_{\pm 24{,}93}$	$1753{,}95_{\pm 43{,}98}$	$1742{,}61_{\pm 33{,}26}$	$3152{,}66_{\pm 96{,}91}$	$5124{,}53_{\pm 62{,}04}$	$3109{,}74_{\pm 129{,}70}$	$3079{,}77_{\pm 185{,}62}$	$4383{,}51_{\pm 217{,}69}$
Problema 3-100Nodos	$1730{,}80_{\pm 40{,}98}$	$1719{,}55_{\pm 28{,}96}$	$1716{,}00_{\pm 29{,}09}$	$3113{,}16_{\pm 108{,}29}$	$5016{,}46_{\pm 43{,}85}$	$2896{,}74_{\pm 125{,}63}$	$2877{,}54_{\pm 133{,}61}$	$4266{,}76_{\pm 129{,}31}$
Problema 4-100Nodos	$1460{,}56_{\pm 16{,}93}$	$1445{,}01_{\pm 25{,}18}$	$1454{,}59_{\pm 17{,}52}$	$2811{,}28_{\pm 88{,}87}$	$4447{,}62_{\pm 78{,}75}$	$2587{,}69_{\pm 135{,}14}$	$2635{,}34_{\pm 104{,}78}$	$3757{,}36_{\pm 146{,}10}$
Problema 5-100Nodos	$1372{,}55_{\pm 19{,}75}$	$1385{,}11_{\pm 30{,}09}$	$1375{,}93_{\pm 28{,}76}$	$2743{,}15_{\pm 109{,}03}$	$4419{,}42_{\pm 65{,}90}$	$2581{,}24_{\pm 120{,}26}$	$2620{,}42_{\pm 159{,}02}$	$3738{,}77_{\pm 248{,}35}$
Problema 1-200Nodos	$3983{,}62_{\pm 131{,}40}$	$3065{,}69_{\pm 37{,}65}$	$3101{,}25_{\pm 52{,}28}$	$6546{,}09_{\pm 138{,}90}$	$10103{,}47_{\pm 77{,}51}$	$5730{,}20_{\pm 277{,}37}$	$5714{,}77_{\pm 228{,}00}$	$8500{,}70_{\pm 363{,}26}$
Problema 2-200Nodos	$3217{,}35_{\pm 151{,}68}$	$2397{,}17_{\pm 49{,}00}$	$2442{,}65_{\pm 50{,}18}$	$5864{,}41_{\pm 100{,}99}$	$9616{,}07_{\pm 88{,}22}$	$5385{,}99_{\pm 319{,}00}$	$5462{,}42_{\pm 258{,}06}$	$8093{,}95_{\pm 350{,}26}$
Problema 3-200Nodos	$4186{,}74_{\pm 230{,}88}$	$2922{,}76_{\pm 66{,}57}$	$2943{,}48_{\pm 34{,}44}$	$6444{,}62_{\pm 121{,}80}$	$10801{,}07_{\pm 63{,}21}$	$6117{,}66_{\pm 258{,}94}$	$6113{,}01_{\pm 382{,}20}$	$9323{,}62_{\pm 587{,}69}$
Problema 4-200Nodos	$4471{,}59_{\pm 237{,}12}$	$3101{,}01_{\pm 62{,}95}$	$3113{,}85_{\pm 42{,}11}$	$6922{,}55_{\pm 922{,}71}$	$11051{,}31_{\pm 54{,}66}$	$6082{,}94_{\pm 215{,}76}$	$6162{,}12_{\pm 355{,}29}$	$9536{,}97_{\pm 627{,}85}$
Problema 5-200Nodos	$3689{,}25_{\pm 150{,}19}$	$2766{,}96_{\pm 72{,}07}$	$2784{,}49_{\pm 52{,}03}$	$6470{,}60_{\pm 79{,}13}$	$10666{,}34_{\pm 75{,}94}$	$6018{,}15_{\pm 240{,}98}$	$6063{,}24_{\pm 290{,}45}$	$8836{,}16_{\pm 337{,}00}$

Tabla 7.2: Media y desviación estándar del valor de aptitud ($\bar{x}_{\pm s}$) para las mejores soluciones encontradas por cada algoritmo en 20 ejecuciones de cada problema con distancias aleatorias. El menor valor medio (el mejor) para cada problema está resaltado en gris.

Distancias aleatorias	EDA	gGA	ssGA	CMAES	DE	ElitistES	NonElitistES	PSO
Problema 1-20Nodos	241,86±5,18	258,74±13,32	263,78±14,59	340,03±22,29	403,18±20,08	357,09±40,58	346,27±53,92	377,01±68,31
Problema 2-20Nodos	214,91±11,32	222,60±10,98	224,06±11,55	319,52±26,31	366,78±17,55	354,92±33,76	359,32±51,26	362,94±45,50
Problema 3-20Nodos	210,50±4,90	216,07±6,53	216,15±12,25	267,91±21,68	307,03±23,14	309,26±49,77	303,72±49,19	316,24±43,23
Problema 4-20Nodos	172,90±10,05	178,05±16,68	172,40±14,28	294,74±34,19	354,69±30,97	267,28±39,16	263,67±24,32	294,74±27,13
Problema 5-20Nodos	161,43±5,66	178,88±13,10	184,09±20,50	311,66±35,31	363,50±34,47	292,25±30,19	312,85±52,63	305,62±43,16
Problema 1-40Nodos	271,58±17,97	287,85±21,29	302,76±26,42	455,13±34,77	907,68±35,41	531,31±52,07	523,35±46,88	709,24±85,06
Problema 2-40Nodos	182,75±5,08	196,55±8,86	199,70±13,52	546,81±33,30	961,71±38,97	532,89±66,28	527,10±61,48	734,71±74,81
Problema 3-40Nodos	298,65±14,44	331,50±17,16	341,14±15,77	601,25±45,32	1008,08±44,92	563,04±62,98	550,86±46,58	744,63±60,87
Problema 4-40Nodos	174,26±7,11	185,16±7,59	183,63±9,40	509,11±30,52	962,72±39,70	483,23±50,98	484,17±63,38	707,98±103,91
Problema 5-40Nodos	270,74±10,00	291,15±17,94	286,98±16,31	576,74±34,94	910,82±45,97	513,39±57,20	511,99±56,08	685,77±83,97
Problema 1-60Nodos	349,32±16,54	401,67±33,18	414,26±41,55	1074,74±79,91	2094,96±63,77	914,18±96,31	902,43±109,20	1565,84±317,56
Problema 2-60Nodos	229,27±9,80	247,99±15,23	248,40±12,42	899,51±79,25	2034,21±56,25	824,22±96,65	831,81±83,31	1559,76±320,93
Problema 3-60Nodos	275,12±11,50	288,37±19,54	288,66±17,54	876,76±48,88	2090,54±58,30	844,39±97,39	820,23±88,55	1509,68±183,21
Problema 4-60Nodos	228,48±10,15	241,70±12,82	241,33±11,20	896,69±82,21	1945,01±45,21	803,16±87,70	794,66±105,41	1485,17±260,20
Problema 5-60Nodos	212,33±6,22	224,96±10,64	224,24±11,09	579,58±41,83	1491,90±39,57	628,61±55,48	646,80±82,87	1085,43±101,51
Problema 1-80Nodos	312,45±8,93	322,20±11,73	324,52±12,46	1025,17±93,17	2613,59±64,29	1002,89±87,13	965,12±104,18	1932,20±144,93
Problema 2-80Nodos	285,91±6,53	301,75±9,95	303,24±18,72	1159,94±65,75	2678,84±80,07	989,62±88,35	1050,99±141,06	1864,73±220,04
Problema 3-80Nodos	291,80±5,82	305,32±12,17	305,60±9,83	804,20±44,22	2307,72±49,13	835,44±77,34	892,42±91,03	1679,06±171,56
Problema 4-80Nodos	347,69±14,34	384,93±20,79	376,70±12,72	1236,65±50,79	2934,28±63,01	1166,41±134,54	1236,17±111,36	2263,97±419,44
Problema 5-80Nodos	321,15±12,80	348,73±15,87	342,75±15,96	1068,62±59,84	2578,79±53,89	1015,96±121,75	975,67±90,84	1814,00±308,47
Problema 1-100Nodos	296,64±7,71	310,30±12,86	304,72±10,50	1142,82±63,21	3243,03±43,57	1137,47±83,83	1184,98±130,79	2177,87±203,51
Problema 2-100Nodos	246,25±7,03	265,22±8,82	262,52±12,58	992,26±63,28	2909,39±68,81	995,25±107,03	1024,73±67,29	2015,25±181,84
Problema 3-100Nodos	349,70±12,31	348,44±12,03	354,21±10,04	1268,05±66,94	3505,83±74,56	1282,11±134,84	1365,61±114,40	2517,75±252,15
Problema 4-100Nodos	267,68±22,57	265,17±9,81	273,49±10,79	1166,87±61,52	3317,99±92,37	1149,12±119,46	1223,47±134,31	2483,98±400,61
Problema 5-100Nodos	342,41±14,30	360,15±16,38	369,71±23,50	1183,28±55,00	3243,94±68,28	1099,98±120,65	1142,22±120,75	2287,92±236,94
Problema 1-200Nodos	1271,16±87,08	452,30±14,76	490,19±17,48	2220,64±98,35	7275,03±136,34	2384,50±184,62	2598,72±198,94	5553,53±855,19
Problema 2-200Nodos	1435,93±103,93	502,99±19,35	530,96±13,75	2376,65±111,63	7532,03±93,78	2521,14±270,30	2528,03±267,82	5469,27±521,27
Problema 3-200Nodos	1371,40±91,35	496,27±23,91	526,49±13,22	2405,00±1146,40	7312,68±124,49	2401,01±217,54	2516,84±215,81	5197,54±628,02
Problema 4-200Nodos	2444,46±151,04	632,74±35,26	685,40±32,87	7093,90±2207,34	8348,16±142,54	2852,08±187,82	2948,95±263,66	7570,55±1091,33
Problema 5-200Nodos	2080,21±146,77	565,23±24,01	614,61±22,30	5118,49±2763,97	8203,31±135,53	3084,57±324,21	2900,78±185,26	7248,20±1381,91

Tabla 7.3: Media y desviación estándar ($\bar{x}_{\pm s}$) **del tiempo de ejecución (en minutos) de cada algoritmo en 20 ejecuciones de cada problema con distancias euclídeas.** El menor tiempo medio para cada problema se resalta en gris.

Distancias euclídeas	EDA	gGA	ssGA	CMAES	DE	ElitistES	NonElitistES	PSO
Problema 1-20Nodos	$2{,}73_{\pm 1{,}03}$	$0{,}24_{\pm 0{,}02}$	$0{,}15_{\pm 0{,}01}$	$0{,}39_{\pm 0{,}02}$	$0{,}99_{\pm 0{,}31}$	$0{,}74_{\pm 0{,}18}$	$0{,}69_{\pm 0{,}29}$	$0{,}36_{\pm 0{,}10}$
Problema 2-20Nodos	$0{,}97_{\pm 0{,}13}$	$0{,}16_{\pm 0{,}00}$	$0{,}13_{\pm 0{,}01}$	$0{,}21_{\pm 0{,}01}$	$0{,}65_{\pm 0{,}22}$	$0{,}31_{\pm 0{,}08}$	$0{,}34_{\pm 0{,}12}$	$0{,}19_{\pm 0{,}05}$
Problema 3-20Nodos	$1{,}70_{\pm 0{,}43}$	$0{,}19_{\pm 0{,}01}$	$0{,}14_{\pm 0{,}01}$	$0{,}32_{\pm 0{,}01}$	$0{,}90_{\pm 0{,}37}$	$0{,}48_{\pm 0{,}18}$	$0{,}50_{\pm 0{,}22}$	$0{,}26_{\pm 0{,}09}$
Problema 4-20Nodos	$1{,}81_{\pm 0{,}25}$	$0{,}22_{\pm 0{,}02}$	$0{,}16_{\pm 0{,}02}$	$0{,}41_{\pm 0{,}02}$	$0{,}79_{\pm 0{,}24}$	$0{,}78_{\pm 0{,}29}$	$0{,}79_{\pm 0{,}28}$	$0{,}31_{\pm 0{,}10}$
Problema 5-20Nodos	$2{,}16_{\pm 0{,}34}$	$0{,}24_{\pm 0{,}03}$	$0{,}16_{\pm 0{,}02}$	$0{,}45_{\pm 0{,}02}$	$0{,}76_{\pm 0{,}25}$	$0{,}76_{\pm 0{,}18}$	$0{,}75_{\pm 0{,}30}$	$0{,}32_{\pm 0{,}08}$
Problema 1-40Nodos	$18{,}42_{\pm 5{,}30}$	$1{,}19_{\pm 0{,}21}$	$0{,}49_{\pm 0{,}07}$	$1{,}97_{\pm 0{,}14}$	$2{,}08_{\pm 0{,}51}$	$2{,}99_{\pm 1{,}27}$	$3{,}12_{\pm 1{,}24}$	$0{,}95_{\pm 0{,}20}$
Problema 2-40Nodos	$7{,}39_{\pm 1{,}56}$	$0{,}89_{\pm 0{,}20}$	$0{,}38_{\pm 0{,}05}$	$1{,}35_{\pm 0{,}07}$	$1{,}81_{\pm 0{,}66}$	$2{,}45_{\pm 0{,}74}$	$2{,}59_{\pm 1{,}06}$	$0{,}76_{\pm 0{,}30}$
Problema 3-40Nodos	$19{,}11_{\pm 5{,}82}$	$1{,}33_{\pm 0{,}31}$	$0{,}48_{\pm 0{,}10}$	$2{,}02_{\pm 0{,}11}$	$1{,}98_{\pm 0{,}72}$	$3{,}52_{\pm 0{,}86}$	$3{,}95_{\pm 1{,}22}$	$0{,}95_{\pm 0{,}31}$
Problema 4-40Nodos	$12{,}64_{\pm 3{,}55}$	$1{,}04_{\pm 0{,}19}$	$0{,}36_{\pm 0{,}07}$	$1{,}66_{\pm 0{,}12}$	$2{,}10_{\pm 0{,}98}$	$2{,}76_{\pm 0{,}89}$	$2{,}94_{\pm 0{,}88}$	$0{,}89_{\pm 0{,}23}$
Problema 5-40Nodos	$11{,}29_{\pm 3{,}00}$	$1{,}04_{\pm 0{,}34}$	$0{,}35_{\pm 0{,}09}$	$1{,}69_{\pm 0{,}10}$	$2{,}00_{\pm 0{,}62}$	$2{,}91_{\pm 0{,}71}$	$2{,}87_{\pm 0{,}67}$	$0{,}78_{\pm 0{,}22}$
Problema 1-60Nodos	$44{,}70_{\pm 11{,}96}$	$2{,}55_{\pm 0{,}66}$	$0{,}76_{\pm 0{,}11}$	$3{,}30_{\pm 0{,}25}$	$2{,}55_{\pm 1{,}02}$	$6{,}70_{\pm 2{,}26}$	$5{,}86_{\pm 1{,}96}$	$1{,}22_{\pm 0{,}41}$
Problema 2-60Nodos	$111{,}04_{\pm 35{,}23}$	$4{,}49_{\pm 1{,}03}$	$1{,}24_{\pm 0{,}23}$	$5{,}27_{\pm 0{,}71}$	$2{,}49_{\pm 0{,}97}$	$8{,}67_{\pm 2{,}78}$	$9{,}24_{\pm 2{,}60}$	$1{,}36_{\pm 0{,}47}$
Problema 3-60Nodos	$58{,}17_{\pm 16{,}22}$	$2{,}40_{\pm 0{,}39}$	$0{,}75_{\pm 0{,}11}$	$3{,}54_{\pm 0{,}30}$	$2{,}25_{\pm 0{,}72}$	$7{,}23_{\pm 2{,}26}$	$7{,}19_{\pm 2{,}33}$	$1{,}24_{\pm 0{,}49}$
Problema 4-60Nodos	$55{,}99_{\pm 16{,}27}$	$3{,}72_{\pm 0{,}85}$	$1{,}17_{\pm 0{,}20}$	$3{,}55_{\pm 0{,}20}$	$2{,}67_{\pm 1{,}13}$	$7{,}32_{\pm 2{,}62}$	$8{,}86_{\pm 3{,}01}$	$1{,}34_{\pm 0{,}50}$
Problema 5-60Nodos	$54{,}10_{\pm 17{,}72}$	$3{,}31_{\pm 0{,}59}$	$1{,}08_{\pm 0{,}17}$	$3{,}75_{\pm 0{,}32}$	$2{,}44_{\pm 0{,}73}$	$8{,}20_{\pm 2{,}53}$	$9{,}18_{\pm 2{,}23}$	$1{,}43_{\pm 0{,}37}$
Problema 1-80Nodos	$367{,}18_{\pm 85{,}91}$	$13{,}65_{\pm 2{,}53}$	$3{,}80_{\pm 0{,}66}$	$7{,}41_{\pm 0{,}58}$	$3{,}86_{\pm 1{,}76}$	$16{,}24_{\pm 5{,}74}$	$15{,}84_{\pm 4{,}80}$	$1{,}95_{\pm 0{,}66}$
Problema 2-80Nodos	$166{,}08_{\pm 30{,}83}$	$5{,}56_{\pm 0{,}91}$	$1{,}48_{\pm 0{,}28}$	$4{,}30_{\pm 0{,}29}$	$3{,}23_{\pm 0{,}90}$	$11{,}19_{\pm 3{,}01}$	$12{,}11_{\pm 3{,}14}$	$1{,}47_{\pm 0{,}63}$
Problema 3-80Nodos	$231{,}12_{\pm 54{,}79}$	$10{,}01_{\pm 2{,}03}$	$2{,}87_{\pm 0{,}42}$	$6{,}00_{\pm 0{,}18}$	$3{,}79_{\pm 1{,}41}$	$15{,}94_{\pm 5{,}36}$	$16{,}35_{\pm 4{,}68}$	$2{,}04_{\pm 0{,}59}$
Problema 4-80Nodos	$138{,}41_{\pm 40{,}87}$	$5{,}89_{\pm 1{,}00}$	$1{,}51_{\pm 0{,}20}$	$4{,}54_{\pm 0{,}27}$	$3{,}39_{\pm 0{,}86}$	$12{,}12_{\pm 3{,}76}$	$10{,}38_{\pm 3{,}65}$	$1{,}84_{\pm 0{,}75}$
Problema 5-80Nodos	$156{,}10_{\pm 50{,}10}$	$6{,}50_{\pm 1{,}19}$	$2{,}29_{\pm 0{,}37}$	$4{,}92_{\pm 0{,}31}$	$4{,}01_{\pm 1{,}26}$	$15{,}49_{\pm 4{,}69}$	$13{,}85_{\pm 5{,}69}$	$1{,}84_{\pm 0{,}78}$
Problema 1-100Nodos	$458{,}15_{\pm 61{,}01}$	$15{,}36_{\pm 3{,}15}$	$4{,}75_{\pm 1{,}07}$	$8{,}01_{\pm 0{,}64}$	$3{,}76_{\pm 1{,}24}$	$17{,}76_{\pm 5{,}67}$	$18{,}99_{\pm 4{,}78}$	$2{,}82_{\pm 1{,}13}$
Problema 2-100Nodos	$667{,}65_{\pm 175{,}45}$	$19{,}02_{\pm 4{,}91}$	$7{,}80_{\pm 1{,}52}$	$10{,}87_{\pm 0{,}65}$	$4{,}77_{\pm 1{,}45}$	$19{,}66_{\pm 6{,}25}$	$20{,}80_{\pm 6{,}88}$	$2{,}78_{\pm 1{,}06}$
Problema 3-100Nodos	$478{,}57_{\pm 122{,}69}$	$15{,}93_{\pm 3{,}82}$	$4{,}86_{\pm 0{,}98}$	$8{,}31_{\pm 0{,}88}$	$4{,}17_{\pm 1{,}40}$	$19{,}36_{\pm 5{,}56}$	$19{,}04_{\pm 5{,}91}$	$2{,}37_{\pm 0{,}87}$
Problema 4-100Nodos	$568{,}66_{\pm 81{,}96}$	$16{,}89_{\pm 3{,}77}$	$5{,}77_{\pm 1{,}12}$	$9{,}12_{\pm 0{,}63}$	$3{,}68_{\pm 1{,}17}$	$21{,}27_{\pm 6{,}93}$	$20{,}02_{\pm 5{,}80}$	$2{,}69_{\pm 0{,}94}$
Problema 5-100Nodos	$756{,}07_{\pm 201{,}91}$	$18{,}17_{\pm 4{,}20}$	$7{,}32_{\pm 1{,}58}$	$10{,}46_{\pm 0{,}74}$	$4{,}13_{\pm 1{,}48}$	$21{,}82_{\pm 7{,}22}$	$20{,}40_{\pm 8{,}42}$	$2{,}55_{\pm 0{,}96}$
Problema 1-200Nodos	$1851{,}28_{\pm 578{,}33}$	$58{,}70_{\pm 6{,}11}$	$52{,}94_{\pm 16{,}15}$	$50{,}42_{\pm 4{,}96}$	$5{,}12_{\pm 1{,}60}$	$41{,}18_{\pm 11{,}47}$	$44{,}84_{\pm 12{,}64}$	$8{,}39_{\pm 2{,}92}$
Problema 2-200Nodos	$2889{,}72_{\pm 672{,}79}$	$66{,}88_{\pm 12{,}41}$	$93{,}88_{\pm 32{,}80}$	$84{,}58_{\pm 7{,}54}$	$8{,}80_{\pm 3{,}14}$	$49{,}89_{\pm 21{,}56}$	$47{,}18_{\pm 14{,}24}$	$10{,}31_{\pm 4{,}06}$
Problema 3-200Nodos	$2549{,}65_{\pm 634{,}00}$	$70{,}65_{\pm 13{,}45}$	$75{,}92_{\pm 15{,}74}$	$66{,}17_{\pm 3{,}47}$	$5{,}82_{\pm 1{,}87}$	$45{,}85_{\pm 16{,}05}$	$47{,}73_{\pm 18{,}19}$	$7{,}76_{\pm 3{,}91}$
Problema 4-200Nodos	$2309{,}51_{\pm 549{,}53}$	$63{,}77_{\pm 10{,}52}$	$60{,}45_{\pm 17{,}78}$	$51{,}61_{\pm 10{,}23}$	$5{,}70_{\pm 1{,}97}$	$51{,}49_{\pm 15{,}17}$	$49{,}43_{\pm 18{,}83}$	$7{,}76_{\pm 3{,}28}$
Problema 5-200Nodos	$2487{,}08_{\pm 498{,}49}$	$68{,}50_{\pm 15{,}06}$	$76{,}44_{\pm 19{,}20}$	$75{,}21_{\pm 6{,}41}$	$6{,}76_{\pm 2{,}56}$	$47{,}86_{\pm 14{,}41}$	$47{,}06_{\pm 15{,}88}$	$10{,}81_{\pm 3{,}87}$

Tabla 7.4: Media y desviación estándar ($\bar{x}_{\pm s}$) del tiempo de ejecución (en minutos) de cada algoritmo en 20 ejecuciones de cada problema con distancias aleatorias. El menor tiempo medio para cada problema se resalta en gris.

Distancias aleatorias	EDA	gGA	ssGA	CMAES	DE	ElitistES	NonElitistES	PSO
Problema 1-20Nodos	$2{,}09_{\pm 0{,}53}$	$0{,}23_{\pm 0{,}02}$	$0{,}14_{\pm 0{,}02}$	$0{,}34_{\pm 0{,}01}$	$0{,}68_{\pm 0{,}26}$	$0{,}60_{\pm 0{,}22}$	$0{,}53_{\pm 0{,}21}$	$0{,}25_{\pm 0{,}10}$
Problema 2-20Nodos	$2{,}22_{\pm 0{,}78}$	$0{,}25_{\pm 0{,}02}$	$0{,}15_{\pm 0{,}01}$	$0{,}42_{\pm 0{,}02}$	$0{,}71_{\pm 0{,}18}$	$0{,}65_{\pm 0{,}23}$	$0{,}57_{\pm 0{,}26}$	$0{,}30_{\pm 0{,}05}$
Problema 3-20Nodos	$1{,}38_{\pm 0{,}34}$	$0{,}17_{\pm 0{,}01}$	$0{,}13_{\pm 0{,}01}$	$0{,}28_{\pm 0{,}01}$	$0{,}66_{\pm 0{,}27}$	$0{,}40_{\pm 0{,}10}$	$0{,}48_{\pm 0{,}16}$	$0{,}22_{\pm 0{,}08}$
Problema 4-20Nodos	$1{,}78_{\pm 0{,}60}$	$0{,}22_{\pm 0{,}02}$	$0{,}14_{\pm 0{,}01}$	$0{,}36_{\pm 0{,}02}$	$0{,}64_{\pm 0{,}24}$	$0{,}56_{\pm 0{,}13}$	$0{,}58_{\pm 0{,}18}$	$0{,}28_{\pm 0{,}10}$
Problema 5-20Nodos	$2{,}20_{\pm 0{,}62}$	$0{,}21_{\pm 0{,}01}$	$0{,}15_{\pm 0{,}02}$	$0{,}38_{\pm 0{,}02}$	$0{,}63_{\pm 0{,}23}$	$0{,}58_{\pm 0{,}17}$	$0{,}58_{\pm 0{,}17}$	$0{,}24_{\pm 0{,}05}$
Problema 1-40Nodos	$8{,}31_{\pm 3{,}89}$	$0{,}72_{\pm 0{,}09}$	$0{,}31_{\pm 0{,}05}$	$1{,}26_{\pm 0{,}05}$	$1{,}74_{\pm 0{,}60}$	$2{,}26_{\pm 0{,}97}$	$2{,}16_{\pm 0{,}83}$	$0{,}76_{\pm 0{,}16}$
Problema 2-40Nodos	$22{,}44_{\pm 7{,}35}$	$1{,}84_{\pm 0{,}41}$	$0{,}59_{\pm 0{,}12}$	$2{,}44_{\pm 0{,}23}$	$2{,}66_{\pm 0{,}84}$	$4{,}44_{\pm 1{,}54}$	$4{,}85_{\pm 1{,}89}$	$0{,}97_{\pm 0{,}30}$
Problema 3-40Nodos	$8{,}08_{\pm 1{,}97}$	$0{,}85_{\pm 0{,}19}$	$0{,}35_{\pm 0{,}06}$	$1{,}34_{\pm 0{,}08}$	$1{,}53_{\pm 0{,}45}$	$2{,}04_{\pm 0{,}57}$	$2{,}30_{\pm 0{,}75}$	$0{,}80_{\pm 0{,}22}$
Problema 4-40Nodos	$24{,}16_{\pm 9{,}12}$	$2{,}06_{\pm 0{,}56}$	$0{,}62_{\pm 0{,}11}$	$2{,}32_{\pm 0{,}17}$	$2{,}36_{\pm 0{,}85}$	$4{,}14_{\pm 1{,}75}$	$4{,}83_{\pm 1{,}61}$	$0{,}94_{\pm 0{,}29}$
Problema 5-40Nodos	$8{,}32_{\pm 2{,}38}$	$0{,}75_{\pm 0{,}11}$	$0{,}28_{\pm 0{,}04}$	$1{,}30_{\pm 0{,}06}$	$1{,}94_{\pm 0{,}63}$	$2{,}28_{\pm 0{,}76}$	$2{,}34_{\pm 0{,}65}$	$0{,}73_{\pm 0{,}20}$
Problema 1-60Nodos	$75{,}52_{\pm 14{,}76}$	$4{,}15_{\pm 1{,}07}$	$1{,}12_{\pm 0{,}22}$	$4{,}22_{\pm 0{,}20}$	$2{,}34_{\pm 0{,}81}$	$7{,}49_{\pm 2{,}83}$	$7{,}85_{\pm 2{,}15}$	$1{,}27_{\pm 0{,}48}$
Problema 2-60Nodos	$153{,}65_{\pm 20{,}60}$	$7{,}06_{\pm 1{,}58}$	$2{,}07_{\pm 0{,}32}$	$5{,}39_{\pm 0{,}45}$	$3{,}16_{\pm 0{,}92}$	$11{,}13_{\pm 3{,}52}$	$11{,}22_{\pm 3{,}66}$	$1{,}58_{\pm 0{,}58}$
Problema 3-60Nodos	$103{,}67_{\pm 21{,}74}$	$6{,}43_{\pm 1{,}69}$	$1{,}54_{\pm 0{,}25}$	$4{,}68_{\pm 0{,}40}$	$2{,}47_{\pm 0{,}60}$	$9{,}49_{\pm 2{,}68}$	$9{,}99_{\pm 3{,}67}$	$1{,}56_{\pm 0{,}61}$
Problema 4-60Nodos	$137{,}21_{\pm 21{,}27}$	$7{,}07_{\pm 1{,}33}$	$1{,}97_{\pm 0{,}38}$	$5{,}11_{\pm 0{,}29}$	$3{,}50_{\pm 1{,}29}$	$10{,}97_{\pm 3{,}54}$	$12{,}06_{\pm 5{,}23}$	$1{,}60_{\pm 0{,}55}$
Problema 5-60Nodos	$83{,}41_{\pm 20{,}38}$	$5{,}52_{\pm 1{,}44}$	$1{,}44_{\pm 0{,}26}$	$4{,}07_{\pm 0{,}28}$	$2{,}66_{\pm 0{,}86}$	$9{,}03_{\pm 2{,}78}$	$9{,}68_{\pm 3{,}71}$	$1{,}34_{\pm 0{,}57}$
Problema 1-80Nodos	$388{,}91_{\pm 65{,}96}$	$13{,}54_{\pm 3{,}91}$	$3{,}73_{\pm 0{,}77}$	$6{,}71_{\pm 0{,}45}$	$3{,}45_{\pm 1{,}16}$	$16{,}18_{\pm 5{,}02}$	$18{,}39_{\pm 5{,}04}$	$1{,}98_{\pm 0{,}56}$
Problema 2-80Nodos	$471{,}47_{\pm 104{,}24}$	$13{,}47_{\pm 2{,}38}$	$4{,}00_{\pm 0{,}77}$	$7{,}50_{\pm 0{,}76}$	$3{,}65_{\pm 1{,}34}$	$18{,}18_{\pm 8{,}13}$	$15{,}40_{\pm 6{,}79}$	$2{,}22_{\pm 0{,}65}$
Problema 3-80Nodos	$198{,}37_{\pm 30{,}49}$	$9{,}77_{\pm 2{,}54}$	$2{,}69_{\pm 0{,}45}$	$5{,}49_{\pm 0{,}55}$	$3{,}49_{\pm 1{,}11}$	$16{,}15_{\pm 5{,}19}$	$13{,}73_{\pm 3{,}79}$	$1{,}85_{\pm 0{,}64}$
Problema 4-80Nodos	$641{,}80_{\pm 139{,}22}$	$16{,}31_{\pm 4{,}45}$	$4{,}34_{\pm 0{,}68}$	$8{,}21_{\pm 0{,}55}$	$3{,}62_{\pm 0{,}99}$	$15{,}84_{\pm 4{,}83}$	$16{,}00_{\pm 7{,}96}$	$2{,}20_{\pm 0{,}86}$
Problema 5-80Nodos	$339{,}20_{\pm 56{,}27}$	$10{,}17_{\pm 2{,}40}$	$2{,}99_{\pm 0{,}69}$	$6{,}18_{\pm 0{,}63}$	$3{,}33_{\pm 1{,}15}$	$15{,}19_{\pm 7{,}02}$	$15{,}33_{\pm 4{,}86}$	$2{,}35_{\pm 0{,}97}$
Problema 1-100Nodos	$928{,}75_{\pm 126{,}28}$	$20{,}28_{\pm 5{,}20}$	$6{,}98_{\pm 1{,}37}$	$8{,}88_{\pm 0{,}51}$	$3{,}89_{\pm 1{,}26}$	$17{,}72_{\pm 5{,}67}$	$18{,}45_{\pm 5{,}07}$	$2{,}58_{\pm 1{,}06}$
Problema 2-100Nodos	$1006{,}34_{\pm 154{,}97}$	$17{,}08_{\pm 3{,}99}$	$7{,}23_{\pm 1{,}64}$	$9{,}40_{\pm 0{,}58}$	$3{,}55_{\pm 0{,}98}$	$21{,}82_{\pm 6{,}51}$	$18{,}97_{\pm 6{,}44}$	$2{,}86_{\pm 1{,}18}$
Problema 3-100Nodos	$1555{,}73_{\pm 291{,}76}$	$22{,}00_{\pm 5{,}08}$	$8{,}09_{\pm 1{,}01}$	$10{,}98_{\pm 0{,}76}$	$3{,}80_{\pm 2{,}02}$	$21{,}86_{\pm 6{,}56}$	$16{,}15_{\pm 5{,}63}$	$2{,}93_{\pm 1{,}01}$
Problema 4-100Nodos	$1668{,}46_{\pm 492{,}78}$	$22{,}61_{\pm 6{,}10}$	$9{,}00_{\pm 2{,}04}$	$12{,}46_{\pm 1{,}16}$	$3{,}16_{\pm 0{,}68}$	$20{,}74_{\pm 6{,}32}$	$18{,}31_{\pm 5{,}92}$	$2{,}87_{\pm 0{,}89}$
Problema 5-100Nodos	$593{,}13_{\pm 129{,}37}$	$18{,}20_{\pm 3{,}01}$	$5{,}09_{\pm 1{,}04}$	$6{,}95_{\pm 0{,}57}$	$3{,}47_{\pm 1{,}28}$	$18{,}77_{\pm 5{,}76}$	$16{,}80_{\pm 5{,}14}$	$2{,}59_{\pm 0{,}94}$
Problema 1-200Nodos	$2146{,}92_{\pm 391{,}88}$	$80{,}90_{\pm 18{,}08}$	$89{,}00_{\pm 21{,}93}$	$66{,}19_{\pm 5{,}46}$	$6{,}68_{\pm 2{,}12}$	$47{,}10_{\pm 14{,}40}$	$42{,}44_{\pm 12{,}85}$	$9{,}59_{\pm 5{,}13}$
Problema 2-200Nodos	$2135{,}35_{\pm 367{,}66}$	$80{,}03_{\pm 18{,}68}$	$82{,}50_{\pm 18{,}49}$	$65{,}59_{\pm 4{,}79}$	$7{,}49_{\pm 2{,}37}$	$47{,}69_{\pm 18{,}21}$	$48{,}55_{\pm 18{,}68}$	$11{,}01_{\pm 4{,}74}$
Problema 3-200Nodos	$2039{,}26_{\pm 344{,}44}$	$83{,}45_{\pm 15{,}47}$	$78{,}53_{\pm 15{,}96}$	$58{,}04_{\pm 11{,}15}$	$7{,}16_{\pm 2{,}75}$	$51{,}06_{\pm 19{,}41}$	$39{,}72_{\pm 14{,}19}$	$10{,}69_{\pm 4{,}27}$
Problema 4-200Nodos	$2239{,}56_{\pm 327{,}61}$	$114{,}55_{\pm 19{,}63}$	$94{,}83_{\pm 17{,}45}$	$32{,}23_{\pm 23{,}98}$	$6{,}17_{\pm 1{,}86}$	$44{,}96_{\pm 13{,}53}$	$39{,}43_{\pm 13{,}77}$	$4{,}50_{\pm 2{,}55}$
Problema 5-200Nodos	$2872{,}08_{\pm 625{,}65}$	$98{,}33_{\pm 16{,}24}$	$103{,}41_{\pm 20{,}04}$	$64{,}04_{\pm 35{,}14}$	$5{,}70_{\pm 1{,}95}$	$40{,}62_{\pm 14{,}33}$	$49{,}47_{\pm 12{,}56}$	$7{,}05_{\pm 5{,}18}$

Tabla 7.5: p-valores obtenidos en la comparación por parejas de los algoritmos utilizando el procedimiento de Bergmann-Hommel para comparaciones múltiples, para la aptitud y el tiempo de ejecución, para distancias euclídeas y distancias aleatorias. Los p-valores $> 0{,}05$ (la igualdad entre los dos algoritmos no se rechaza) se muestran en negrita.

	Aptitud		Tiempo de ejecución	
	D. Euclídeas	D. Aleatorias	D. Euclídeas	D. Aleatorias
	p-valor	p-valor	p-valor	p-valor
1 EDA vs gGA	**1,000000**	**1,000000**	1,76E-08	1,22E-06
2 EDA vs ssGA	**1,000000**	**0,641832**	1,77E-17	4,51E-16
3 EDA vs CMAES	2,73E-07	3,00E-08	1,51E-07	1,27E-08
4 EDA vs DE	1,10E-19	2,49E-23	1,22E-11	8,29E-12
5 EDA vs ElitistES	3,82E-06	1,22E-06	0,00552	0,00164
6 EDA vs NonElitistES	3,12E-06	1,20E-07	0,00552	0,00403
7 EDA vs PSO	1,11E-15	4,52E-17	8,64E-21	1,07E-21
8 gGA vs ssGA	**1,000000**	**1,000000**	0,03130	0,01096
9 gGA vs CMAES	6,25E-07	6,08E-06	**1,00000**	**0,85839**
10 gGA vs DE	6,05E-19	3,69E-19	**0,80514**	**0,28890**
11 gGA vs ElitistES	8,56E-06	9,29E-05	**0,06471**	**0,34154**
12 gGA vs NonElitistES	6,54E-06	1,89E-05	**0,06471**	**0,32543**
13 gGA vs PSO	4,66E-15	1,40E-13	0,00249	5,89E-05
14 ssGA vs CMAES	3,63E-05	1,24E-04	0,01096	**0,10624**
15 ssGA vs DE	1,11E-15	1,38E-16	**0,27507**	**0,61771**
16 ssGA vs ElitistES	2,76E-04	0,001381	6,33E-07	2,31E-05
17 ssGA vs NonElitistES	2,76E-04	2,47E-04	6,33E-07	8,30E-06
18 ssGA vs PSO	2,61E-12	1,80E-11	**1,00000**	**0,46419**
19 CMAES vs DE	0,001035	1,37E-04	**0,70008**	**0,61771**
20 CMAES vs ElitistES	**1,000000**	**1,000000**	**0,10197**	**0,07082**
21 CMAES vs NonElitistES	**1,000000**	**1,000000**	**0,10197**	**0,06471**
22 CMAES vs PSO	0,030660	0,030660	8,04E-04	0,00133
23 DE vs ElitistES	2,01E-04	9,44E-06	0,00163	0,00403
24 DE vs NonElitistES	2,01E-04	5,24E-05	0,00163	0,00201
25 DE vs PSO	**1,000000**	**0,641832**	**0,06847**	**0,06471**
26 ElitistES vs NonElitistES	**1,000000**	**1,000000**	**1,00000**	**0,85839**
27 ElitistES vs PSO	0,009759	0,005517	5,71E-09	1,12E-08
28 NonElitistES vs PSO	0,009759	0,015979	5,33E-09	2,68E-09

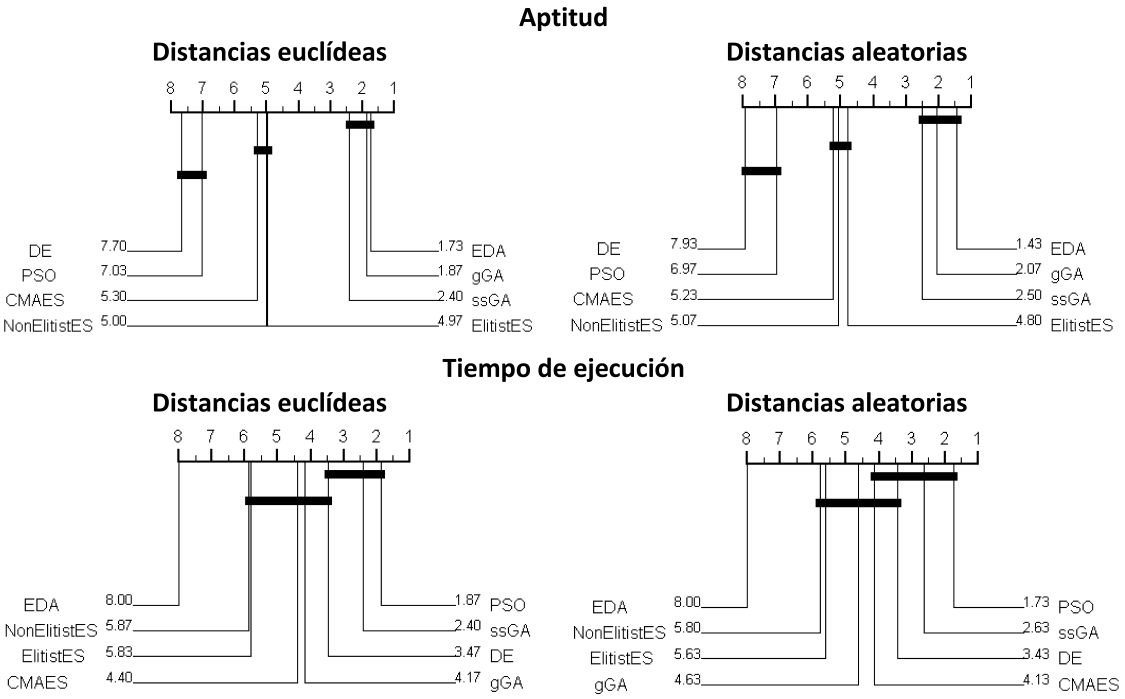


Figura 7.6: Comparación de los ocho algoritmos utilizando el test de Friedman y el procedimiento de Bergmann-Hommel para comparaciones múltiples. Los grupos de algoritmos que no son significativamente diferentes (p-valor $> 0{,}05$) están conectados. Las posiciones menores (las mejores) están a la derecha por lo que los algoritmos en la parte derecha pueden considerarse mejores. La fila superior muestra los diagramas para la aptitud para distancias euclídeas (izquierda) y distancias aleatorias (derecha) y la fila inferior muestra los diagramas para el tiempo de ejecución, también para distancias euclídeas (izquierda) y distancias aleatorias (derecha).

8

Conclusiones y líneas futuras

En este trabajo se ha presentado una novedosa representación basada en permutaciones para resolver una nueva variante del problema DCMST, que hemos introducido por primera vez en la literatura, y hemos llamado el problema DRCMST. Un DRCMST es un DCMST con restricciones adicionales que determinan el rol de los nodos en el árbol (nodos raíz, nodos intermedios o nodos hoja). Establecer las funciones de los nodos puede ser útil en algunos problemas tales como el diseño de redes. En la mayoría de investigaciones sobre DCMST se obtiene un solo árbol. Nosotros aumentamos la flexibilidad del problema al no limitar el número de nodos raíz a uno construyendo, por lo general, bosques de DRCMSTs. Además, en este trabajo se han desarrollado dos demostraciones teóricas sobre la factibilidad del problema DRCMST y el número de árboles en un bosque.

Utilizamos técnicas metaheurísticas para aproximar la solución del problema debido a que el problema DRCMST es NP-complejo. En concreto, se optó por utilizar una gama de diferentes técnicas de computación evolutiva para poder comparar los resultados. Utilizando la representación propuesta, se resolvieron una amplia gama de problemas simulados. Los resultados mostraron que EDA y los algoritmos genéticos (gGA y ssGA) encontraron las mejores soluciones, pero ssGA lo hizo en un tiempo significativamente menor.

La principal ventaja de nuestra representación es que puede codificar más de un árbol simultáneamente. Además, la restricción de grado puede ser diferente para cada nodo. Otro punto fuerte es que es fácil agregar restricciones relacionadas con un problema específico. Por ejemplo, si dos nodos no se pueden conectar en el planteamiento del problema, entonces, un número específico estará prohibido en una posición específica de la permutación.

Probablemente el punto más débil de la representación propuesta es que codi-

fica individuos no válidos (ciclos). Los ciclos de longitud igual a uno son fáciles de detectar y, por tanto, evitar (esta es la causa del mayor número de individuos no válidos). Sin embargo, la permutación debe ser decodificada para detectar la existencia de ciclos de longitud mayor que uno. Por otra parte, diferentes permutaciones pueden codificar el mismo bosque. Nosotros decidimos eliminar esta redundancia seleccionando un individuo representativo entre los individuos redundantes para no aumentar el tamaño del espacio de búsqueda innecesariamente.

Otros aspectos podrían tenerse en cuenta, tales como considerar una función de evaluación de la aptitud más completa. Por ejemplo, si la red está diseñada para la transmisión de señales desde nodos servidor a nodos hoja, entonces, las distancias de los nodos raíz a los nodos hoja deben ser tan cortas como sea posible, ya que las distancias están estrechamente relacionados con el tiempo de transmisión. En este caso, además de reducir al mínimo el coste total (distancia) del bosque resultante, podría ser también beneficioso reducir al mínimo las distancias entre las raíces y las hojas. Si hay varios criterios de optimización a considerar, también podríamos pensar en la conveniencia de optimizar un problema de un solo objetivo (por ejemplo, ponderando los diferentes objetivos) o avanzar hacia un problema multi-objetivo.

Respecto a la redundancia en la representación, Rothlauf (2006), por ejemplo, muestra que las representaciones redundantes parecen mejorar el desempeño de un algoritmo evolutivo. Por ello, un trabajo futuro podría ser analizar la proporción de individuos redundantes que genera nuestra representación en la población, y estudiar si la convergencia de los algoritmos evolutivos considerados mejora al introducir cierto grado de redundancia y cual sería el nivel más adecuado.

Otro aspecto a considerar en un trabajo futuro es la resolución de problemas con un número extremadamente grande de nodos. En este caso, podría ser útil descomponer el problema original en subproblemas de tamaño más pequeño y paralelizar su resolución.

Los algoritmos genéticos se han utilizado ampliamente para resolver el problema DCMST, y algunos autores también utilizan representación de permutaciones para encontrar el DCMST (Krishnamoorthy et al., 2001). Aquí se ha visto que los algoritmos genéticos también se comportan satisfactoriamente en el problema DRCMST. Sin embargo, los EDAs no se han desarrollado ampliamente para problemas de optimización basados en permutaciones. Nosotros añadimos esta estrategia a jMetal y encontramos que también tienen un buen desempeño. En este trabajo se ha utilizado un sencillo EDA univariante (Tsutsui, 2006) por lo que podría merecer la pena examinar EDAs que proporcionen modelos más complejos, capturando relaciones de orden superior entre variables y analizar su rendimiento en el problema DRCMST.

Actualmente estamos trabajando en la elaboración de un artículo con el contenido presentado en esta tesis fin de máster. Una versión inicial de nuestra representación basada en permutaciones, donde se consideraba la construcción de bosques de expansión mínimos con árboles binarios aplicado al estudio del cableado neuronal (Anton-Sanchez et al., 2013), recibió el primer premio en el *workshop* de estudiantes en la conferencia *Genetic and Evolutionary Computation Conference* (GECCO) 2013.

8. CONCLUSIONES Y LÍNEAS FUTURAS

Bibliografía

J.A. Aledo, J.A. Gámez, y D. Molina. Tackling the rank aggregation problem with evolutionary algorithms. *Applied Mathematics and Computation*, 222:632–644, 2013. 25

L. Anton-Sanchez, C. Bielza, y P. Larrañaga. Towards optimal neuronal wiring through estimation of distribution algorithms. In *Proceedings of the Fifteenth Annual Conference on Genetic and Evolutionary Computation*, GECCO '13 Companion, pages 1647–1650, 2013. 11, 49

J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *Journal on Computing*, 6(2):154–160, 1994. 25

B. Bergmann y G. Hommel. Improvements of general multiple test procedures for redundant systems of hypotheses. In *Multiple Hypotheses Testing*, volume 70 of *Medizinische Informatik und Statistik*, pages 100–115. Springer Berlin Heidelberg, 1988. 37

C. Bielza, J.A. Fernández del Pozo, P. Larrañaga, y E. Bengoetxea. Multidimensional statistical analysis of the parameterization of a genetic algorithm for the optimal ordering of tables. *Expert Systems with Applications*, 37(1):804–815, 2010. 25

J. Ceberio, A. Mendiburu, y J.A. Lozano. Introducing the Mallows model on estimation of distribution algorithms. In *Neural Information Processing*, volume 7063 of *Lecture Notes in Computer Science*, pages 461–470. Springer Berlin Heidelberg, 2011. 25

J. Ceberio, E. Irurozki, A. Mendiburu, y J.A. Lozano. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1(1):103–117, 2012. 25

J. Ceberio, E. Irurozki, A. Mendiburu, y J.A. Lozano. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18(2):286–300, 2014. 25

H. Cobb y J. Grefenstette. Genetic algorithms for tracking changing environments. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 523–530. Morgan Kaufmann, 1993. 15

BIBLIOGRAFÍA

M.M. Czajko y J. Wojciechowski. Tree-based access network design under requirements for an aggregation network. *Elektronika - Konstrukcje, Technologie, Zastosowania*, 4: 23–27, 2009. 3

C. Darwin. *On the Origin of Species by Means of Natural Selection.* Murray, London, 1859. 15

A.C.B. Delbem, A. de Carvalho, C.A. Policastro, A.K.O. Pinto, K. Honda, y A.C. García. Node-depth encoding for evolutionary algorithms applied to network design. In *Genetic and Evolutionary Computation*, volume 3102 of *Lecture Notes in Computer Science*, pages 678–687. Springer Berlin Heidelberg, 2004. 2

A.C.B. Delbem, T.W. de Lima, y G.P. Telles. Efficient forest data structure for evolutionary algorithms applied to network design. *IEEE Transactions of Evolutionary Computation*, 16(6):829–846, 2012. 2

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. 38

J.J. Durillo y A.J. Nebro. jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011. 25

M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. 37

M.R. Garey y D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979. 2

N. Hansen y A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 312–317. Morgan Kaufmann, 1996. 15

J. Kennedy y R.C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948. IEEE, 1995. 15

J. Knowles, D. Corne, y M. Oates. A new evolutionary approach to the degree constrained minimum spanning tree problem. *IEEE Transactions on Evolutionary Computation*, 4: 125–134, 2000. 2

M. Krishnamoorthy, A.T. Ernst, y Y.M. Sharaiha. Comparison of algorithms for the degree constrained minimum spanning tree. *Journal of Heuristics*, 7(6):587–611, 2001. 1, 2, 48

J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. 1

P. Larrañaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, y S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13(2):129–170, 1999. 25

P. Larrañaga y J.A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, 2002. 15

R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36:1389–1401, 1957. 1

G.R. Raidl y B.A. Julstrom. Edge sets: An effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3):225–239, 2003. 2

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973. 15

C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22(1):5–13, 1995. 25

F. Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer-Verlag, 2nd edition, 2006. 48

R. Ruiz y C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494, 2005. 25

R. Sedgewick y K. Wayne. *Algorithms*. Addison-Wesley, 4th edition, 2011. 13, 27

S. Soak, D. Corne, y B. Ahn. A new encoding for the degree constrained minimum spanning tree problem. In *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3213 of *Lecture Notes in Computer Science*, pages 952–958. Springer Berlin Heidelberg, 2004. 2

R. Storn y K. Price. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. 15

G. Syswerda. A study of reproduction in generational and steady-state genetic algorithms. *Foundation of Genetic Algorithms*, pages 94–101, 1991. 15

S. Tsutsui. Node histogram vs. edge histogram: A comparison of probabilistic model-building genetic algorithms in permutation domains. In *IEEE Congress on Evolutionary Computation, 2006*, pages 1939–1946, 2006. 26, 48