Universidad Nacional de San Luis

TESIS DE DOCTORADO
Doctorado en Ciencias de la Computación

Optimización Mono y Multiobjetivo a través de una Heurística de Inteligencia Colectiva

Mg. Leticia Cecilia Cagnina

Lic. Susana C. Esquivel
Co-Directora

Dr. Carlos A. Coello Coello
Director

San Luis, Argentina.
Abril de 2010.

Tesis Doctoral:

Optimización Mono y Multiobjetivo a través de una Heurística de Inteligencia Colectiva

Autora: ***Mg. Leticia Cecilia Cagnina***

Grado al que se aspira: ***Doctora en Ciencias de la Computación***

Co-Directora: ***Lic. Susana Cecilia Esquivel***

Director: ***Dr. Carlos Artemio Coello Coello***

Universidad Nacional de San Luis
San Luis, Argentina
Abril de 2010

Dedicatoria

A mamá y papá...

Agradecimientos

A mi co-directoria Lic. Susana Esquivel por su valiosa ayuda y motivación en todo momento,

a mi director Dr. Carlos Coello Coello por su constante orientación y tiempo dedicado,

a mis amigos y compañeros por permitirme compartir todos los buenos momentos generados con este trabajo (y los malos también!),

a mi papá y a mi hermana por estar siempre para lo que sea...

Leticia.

Resumen

La optimización es un tema clave dentro de las Ciencias de la Computación, específicamente la optimización de problemas del mundo real, ya que éstos son clasificados generalmente como "duros" según la teoría de Complejidad Computacional. Estos problemas tienen la característica de que en ellos no se puede garantizar encontrar la mejor solución, en un tiempo razonablemente corto.
Existe una gran variedad de métodos utilizados para la resolución de problemas de optimización duros, pero las *heurísticas* suelen ser más adecuadas debido a que: son simples de implementar, son suficientemente generales como para permitir su adaptación a muchos problemas y, se ha probado exitosamente su aplicación para resolver diferentes tipos de problemas.
Las heurísticas son métodos aproximados de resolución, que utilizan un proceso iterativo para guiar la búsqueda de soluciones, combinando inteligentemente diferentes conceptos derivados de campos tales como: Inteligencia Artificial, Evolución Biológica e Inteligencia Colectiva, entre otros.
Una de las heurísticas perteneciente al paradigma de la Inteligencia Colectiva es *Particle Swarm Optimization* o PSO. Este método de optimización se basa en una población de individuos que recorren el espacio de búsqueda del problema, para encontrar buenas soluciones. PSO ha probado su eficacia en la resolución de distintos tipos de problemas, por lo cual se le ha seleccionado para este trabajo.
En esta tesis se proponen varios algoritmos basados en PSO para resolver problemas mono-objetivo con y sin restricciones (en los cuales se debe optimizar sólo una función objetivo). También se presenta una hibridización de una eficaz técnica matemática con una versión de PSO, para resolver problemas multiobjetivo complejos (en los cuales existen varias funciones objetivo a optimizar).
Los algoritmos propuestos se validaron con funciones estándar de la literatura especializada (benchmarks), y con problemas reales. Éstos últimos son de gran importancia ya que requieren no sólo buenas soluciones, sino que éstas deben generarse en tiempos relativamente cortos.
Los resultados obtenidos en todos los experimentos realizados, se compararon con los de algoritmos representativos del estado del arte en optimización mono y multiobjetivo, demostrando que los algoritmos propuestos son competitivos y, en algunos casos, superiores en desempeño.

Abstract

Optimization is a very important topic within Computer Science. It is particularly important the optimization of real-world problems, since many of them are classified as "hard" according to computational complexity theory. As such, we cannot guarantee to find the best possible solution to them in a reasonably short time.

There are a wide variety of methods that are used to solve hard optimization problems, but *heuristics* tend to be the most appropriate because: they are easy to implement, they are sufficiently general as to allow their adaptation to a wide variety of problems and, they have been found to be successful to solve different types of problems.

Heuristics are approximate solution methods which use an iterative process to guide the search of solutions, combining in a smart way, different concepts derived from fields such as: artificial intelligence, biological evolution, and swarm intelligence, among others.

One of the heuristics belonging to the swarm intelligence paradigm is *Particle Swarm Optimization* or PSO. This optimization method is based on a population of individuals that traverse the search space of a problem aiming to find good solutions. PSO has been found to be effective in the solution of different types of problems, which is the reason why we chose it for this work.

In this thesis, we propose several PSO-based algorithms to solve single-objective problems (i.e., in which we wish to optimize a single objective function) with and without constraints. We also present a hybrid between an effective mathematical programming technique and a PSO-based algorithm, which is designed to solve complex multi-objective optimization problems (i.e., problems having several objective functions to be optimized).

The proposed algorithms were validated using standard test functions taken from the specialized literature (*benchmarks*) and with real-world problems. The latter are of utmost importance since they require not only of high-quality solutions, but they must also be generated in a relatively short time.

The results obtained for all the cases under study were compared with respect to those produced by algorithms representative of the state-of-the-art in either single- or multi-objective optimization, showing that the proposed algorithms are competitive and, in some cases, higher in performance.

Índice general

Índice de Tablas

Índice de Figuras

Índice de Algoritmos

Capítulo 1

Introducción

En este capítulo se describen los objetivos particulares de la presente tesis doctoral, así como los aportes que en la misma se realizan. También se muestra la estructura del informe, detallando el contenido de cada capítulo. Finalmente, se listan las publicaciones derivadas de esta tesis.

Contenido del Capítulo

La palabra *optimización* se utiliza en el ámbito de las ciencias de la computación, para hacer referencia al proceso de encontrar la mejor solución a un problema, dado un conjunto de requerimientos. Las características y requerimientos del problema definen si la mejor solución puede ser determinada, o si generar una solución que satisfaga los requerimientos, es suficiente.

El mundo real está repleto de situaciones que involucran problemas de optimización, las cuales pueden ser de diversos tipos y generarse en diferentes sectores, por ejemplo:

- *Sector de Diseño*: determinación del diseño óptimo de armaduras, de piezas mecánicas, alas de aviones, circuitos, antenas de comunicaciones.
- *Sector de Manufactura*: determinación óptima de una cartera de inversiones, determinación de tandas de producción según requerimientos del mercado.
- *Sector Agrícola*: planeación de siembras, riegos y cosechas.
- *Sector Salud*: optimización de turnos de trabajos de enfermeras y médicos en los hospitales, asignación óptima de los pabellones quirúrgicos, asignación óptima de camas en los hospitales, ruteamiento de las ambulancias en una ciudad.
- *Sector Transporte*: determinación de las rutas óptimas de las líneas de colectivos, minimización de los tiempos de espera en las estaciones, definición de los trazados de las líneas del transporte subterráneo, asignación óptima de trenes de carga, determinación de la mejor ruta con cambios de medios de transporte entre dos puntos de la región, determinación de la ruta óptima cuando hay cambios de medios de transporte: marítimos, terrestres y aéreos.
- *Sector Telecomunicaciones*: optimización de la mensajería a través de plantas telefónicas, determinación de la localización óptima de antenas, ruteamiento de las llamadas entre plantas telefónicas y medios electrónicos en general, localización de concentradores en redes de computadoras, ruteamiento de paquetes de datos en una red de computadoras.
- *Sector Energía*: generación y distribución óptima de la energía eléctrica, localización de plantas de energía eléctrica, configuración de redes de energía eléctrica para absorber el abastecimiento de la población, trazado óptimo de las redes de gas urbano, determinación óptima de la compra de petróleo crudo para producir combustible.
- *Sector Educación*: asignación de los horarios y cursos a las salas de clases, asignación de los profesores a los cursos de un colegio o una institución educativa.

A través de los años se ha demostrado que no existe un único método capaz de resolver eficientemente cualquier problema de optimización, bajo cualquier circunstancia. En general, la selección del método que mejor resuelve un problema está sujeto a la determinación de la complejidad computacional que demande, recursos que insuma, propiedades de convergencia y robustez del método.

Las técnicas de optimización más conocidas son: las exactas, de exploración dirigida, heurísticas de búsqueda local, espectrales, y algoritmos bioinspirados (entre los que se destacan los algoritmos evolutivos).
Los algoritmos bioinspirados son aquellos que emulan algún comportamiento biológico o de la naturaleza para realizar una búsqueda en el espacio de posibles soluciones.
Uno de los algoritmos bioinspirados más destacados es el *Particle Swarm Optimization* (optimización mediante cúmulo de partículas), y es el que se seleccionó para trabajar en la presente tesis.
El algoritmo *Particle Swarm Optimization* o PSO es un algoritmo de búsqueda poblacional que intenta simular el comportamiento social de las aves de una bandada. La motivación de esta heurística fue reproducir los patrones de vuelo que producen que las aves vuelen sincronizadamente, así como la reagrupación óptima que experimentan cuando cambian la dirección de vuelo y, principalmente, imitar cómo la bandada converge a un lugar preciso cuando una de las aves divisa alimento.
Los resultados obtenidos con la heurística PSO fueron, desde un inicio, muy alentadores. Por ello, en el presente trabajo de tesis, se utiliza esta heurística como base para la propuesta de nuevos algoritmos que permitan resolver diversos problemas.

1.1. Objetivo de la tesis

El objetivo primordial de este trabajo, es la propuesta de la varios algoritmos PSO para tratar problemas de optimización mono-objetivo con y sin restricciones, y también problemas multiobjetivo complejos. Aunque existen técnicas que resuelven esos problemas eficientemente, este trabajo se concentra en el desarrollo de algoritmos PSO que cubran los siguientes aspectos:

- Eficiencia, en cuanto a la calidad de soluciones.
- Complejidad computacional, con el objetivo de realizar algoritmos relativamente sencillos que no insuman recursos de forma indiscriminada.
- Tiempo de respuesta, un aspecto fundamentalmente importante en la resolución de problemas de mundo real.

Para validar los algoritmos desarrollados se utilizaron los siguientes problemas de optimización: un benchmark de 13 funciones sin restricciones, un benchmark de 24 funciones con restricciones, diseño óptimo de armaduras, diseño ingenieril de piezas mecánicas, despacho de cargas eléctricas en sistemas de potencia y, problemas multiobjetivo complejos con dos y tres funciones objetivo.
Para todos los casos, se compararon los resultados con los obtenidos con algoritmos representativos del estado del arte en optimización mono y multiobjetivo.

1.2. Aportes al lector

Este trabajo pretende realizar los siguientes aportes al lector:

- Un breve resumen de las diferentes técnicas existentes para la resolución de problemas generales mono-objetivo y multiobjetivo.
- Un estudio completo de la heurística PSO: detalle de componentes, características de búsqueda, diferentes tipos de variantes existentes y aplicaciones.
- Se proponen fórmulas novedosas para la actualización de las partículas y sus velocidades, destacando las principales ventajas de los modelos local y global de PSO.
- Se introduce la utilización de dos cúmulos como población del método, las que se implementan de manera distinta a la de otros algoritmos basados en PSO.
- Se presenta un mecanismo de movimiento de partículas con la finalidad de evitar convergencia prematura, particularmente en problemas de optimización con restricciones.
- Se propone un operador de mutación dinámica para forzar al método a explorar nuevas zonas del espacio de búsqueda.
- Se introduce la hibridización de una técnica matemática con un algoritmo basado en PSO, para resolver efectivamente problemas de optimización multiobjetivo complejos.

1.3. Organización del informe

La tesis está organizada de la siguiente manera.

Capítulo 1: **"Introducción".** El presente capítulo.

Capítulo 2: **"Problemas de Optimización y Métodos de Resolución".** Se describen las técnicas de resolución de problemas más utilizadas en el área de optimización en general.

Capítulo 3: **"La Heurística Particle Swarm Optimization".** Se explica la estructura básica de la heurística, así como sus componentes, diferentes variantes, un breve estado del arte y, una comparación entre PSO y Computación Evolutiva.

Capítulo 4: **"Solución de Problemas Mono-objetivo sin restricciones".** Se propone un algoritmo basado en PSO para problemas de optimización sin restricciones, el cual fue validado con un benchmark de 13 funciones con diversas características.

Capítulo 5: **"Solución de Problemas Mono-objetivo con restricciones".** Se propone un algoritmo basado en PSO para resolver problemas con restricciones, el cual es una versión mejorada del algoritmo presentado en el capítulo anterior. Esta versión fue validada con cuatro casos de prueba: un benchmark de 24 funciones con restricciones, diseño óptimo de armaduras, diseño ingenieril de piezas mecánicas y, despacho de cargas

eléctricas en sistemas de potencia.

Capítulo 6: **"Solución de Problemas Multiobjetivo.** Se propone una hibridización de la técnica *epsilon-constraint* con un algoritmo basado en PSO para resolver problemas complejos de optimización multiobjetivo.

Capítulo 7: **"Análisis de parámetros de PSO".** Se describe el análisis estadístico realizado para determinar el conjunto de valores de parámetros para los algoritmos propuestos.

Capítulo 8: **"Conclusiones y Trabajo Futuro".** Se detallan las conclusiones generales de todo el trabajo realizado, y se enuncian algunos lineamientos para futuras extensiones.

Apéndice A: **"Funciones mono-objetivo sin restricciones con 30 variables de decisión".** Se describen las 13 funciones utilizadas en el Capítulo 4.

Apéndice B: **"Funciones mono-objetivo con restricciones".** Se describen las 24 funciones del benchmark, utilizadas en el Capítulo 5.

Apéndice C: **" Funciones de diseño óptimo de armaduras".** Se describen los tres problemas de armaduras utilizados en el Capítulo 5.

Apéndice D: **"Funciones de diseño ingenieril de piezas".** Se describen cuatro problemas de diseño de piezas, utilizados en el Capítulo 5.

Apéndice E: **"Problema de despacho de cargas eléctricas".** Se describen los tres casos del problema en cuestión, mostrando los datos de entrada requeridos por cada uno. Estos problemas fueron utilizados en el Capítulo 5.

Apéndice F: **"Funciones multiobjetivo".** Se describen los problemas multiobjetivo utilizados en el Capítulo 6.

Apéndice G: **" Resultados de LHD para las 20 configuraciones".** Se muestran los valores obtenidos con cada configuración de parámetros para las 13 funciones sin restricciones, las 24 funciones con restricciones y los problemas de diseño óptimo de armaduras (utilizados en el Capítulo 7).

1.4. Lista de publicaciones derivadas

1.4.1. Capítulos en libros

Particle Swarm Optimization for clustering short-text corpora (Diego Ingaramo, Marcelo Errecalde, Leticia Cagnina y Paolo Rosso), capítulo incluido en el libro *Computational Intelligence and Bioengineering (Essays in Memory of Antonina Starita)*, subseries: KBIES (FAIA-Knowledge-Based Intelligent Engineering Systems), series: "Frontiers in Artificial Intelligence and Applications". Editores: Francesco Masulli, Alessio Micheli, Alessandro Sperduti. Págs. 3-19. IOS Press 2009. ISBN: 978-1-60750-010-0. ISSN: 0922-6389.

A Particle Swarm Optimizer for Constrained Numerical Optimization (Leticia C. Cagnina, Susana C. Esquivel y Carlos A. Coello-Coello), en *9th International Conference - Parallel Problem Solving from Nature (PPSN IX), Lecture Notes in Computer Science 4193*. Editores: Thomas Philip Runarsson, Hans-Georg Beyer, Edmund K. Burke, Juan J. Merelo Guervós, L. Darrell Whitley y Xin Yao. Págs 910-919. Springer 2006, ISBN: 3-540-38990-3.

1.4.2. Revistas

A Fast Particle Swarm Algorithm For Solving Smooth and Non-smooth Economic Dispatch Problems (Leticia C. Cagnina, Susana C. Esquivel y Carlos A. Coello Coello), enviado el 15/11/09 a *Engineering Optimization*. ISSN: 0305-215x. Taylor & Francis. ID del manuscrito: GENO-2009-0288. En prensa.

Solving Constrained Optimization Problems with a Particle Swarm Optimization Algorithm (Leticia C. Cagnina, Susana C. Esquivel y Carlos A. Coello Coello), enviado el 01/08/09 a *Engineering Optimization*. ISSN: 0305-215x. Taylor & Francis. ID del manuscrito: GENO-2009-0187. En revisión.

Solving Hard Multiobjective Problems with a Hybridized Method (Leticia C. Cagnina y Susana C. Esquivel), en *Journal of Computer Science and Technology (JCS&T), edición especial: XV Congreso Argentino en Ciencias de la Computación (CACIC 2009)*. En prensa.

Un nuevo paradigma computacional basado en una antigua investigación biológica (Leticia Cecilia Cagnina), en *Fundamentos en Humanidades*. Directores-Editores: Jorge Ricardo Rodriguez, Carlos Francisco Mazzola y Ramón Sanz Ferramola. ISSN: 1515-4467. ISSN (en línea): 1668-7116. Tirada semestral. Publica: Facultad de Ciencias Humanas, Universidad Nacional de San Luis. En prensa.

Solving Engineering Optimization Problems with the Simple Constrained PSO (Leticia C. Cagnina, Susana C. Esquivel y Carlos A. Coello Coello), en *Informatica: International Journal of Computing and Informatics* Vol. 32 (2008), número 3. Octubre de

2008. Págs. 319-326. ISSN 0350-5596.

A Particle Swarm Optimizer for Multi-Objective Optimization (Leticia Cagnina, Susana Esquivel y Carlos A. Coello Coello), en *Journal of Computer Science and Technology (JCS&T)*, Vol. 5 número 4. Diciembre de 2005. Págs. 204-210. ISSN 1666-6038.

1.4.3. Actas en Congresos Internacionales

A Discrete Particle Swarm Optimizer for Clustering Short-Text Corpora (Leticia C. Cagnina, Marcelo L. Errecalde, Diego A. Ingaramo y Paolo Rosso), en *The 3rd International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2008)*. 13-14 de Octubre 2008 en Ljubljana, Eslovenia. Publicado en las correspondientes memorias, Págs 93-103. (ISBN 978-961-264-002-6).

A Bi-population PSO with a Shake-Mechanism for Solving Constrained Numerical Optimization (Leticia C. Cagnina, Susana C. Esquivel y Carlos A. Coello Coello), en *Congress on Evolutionary Computation (CEC 2007)*. 25 al 28 de Setiembre de 2007 en Singapur. Publicado en las correspondientes memorias, Págs. 670-676. (ISBN: 1-4244-1340-0).

1.4.4. Actas en Congresos y Workshops Nacionales

Optimización de Funciones Mono-Objetivo con y sin restricciones, y Funciones Multi-Objetivo a través de Heurísticas Bio-Inspiradas (Victoria Aragón, Leticia Cagnina y Susana Esquivel), en *XI Workshop de Investigadores en Ciencias de la Computación (WICC 2009)*, en la Universidad Nacional de San Juan, San Juan, Argentina. 7 y 8 de Mayo de 2009. Publicado en las correspondientes memorias (ISBN 978-950-605-570-7). Págs. 163-167.

Multi-Objective Optimization with a Gaussian PSO algorithm (Susana C. Esquivel y Leticia C. Cagnina), en *XIV Congreso Argentino en Ciencias de la Computación (CACIC 2008)*. Realizado en la ciudad de Chilecito, La Rioja, Argentina desde el 7 al 10 de Octubre de 2008. Publicado en las correspondientes memorias. (ISBN: 978-987-24611-0-2). Págs 401-412.

Global Numerical Optimization with a bi-population Particle Swarm Optimizer (Susana C. Esquivel y Leticia C. Cagnina), en *XIII Congreso Argentino en Ciencias de la Computación (CACIC 2007)*. Realizado en Corrientes, Argentina desde el 1 al 5 de Octubre de 2007. Publicado en las correspondientes memorias (ISBN 978-950-656-109-3). Págs 1452-1463.

Metaheurísticas basadas en Inteligencia Computacional Aplicadas a la Resolución de Problemas de Optimización Numérica con y sin Restricciones y Optimización Combinatoria (Victoria Aragón, Leticia Cagnina, Claudia Gatica y Susana Esquivel), en *IX Workshop de Investigadores en Ciencias de la Computación (WICC 2007)*, en la Universidad Nacional de la Patagonia San Juan Bosco, Trelew, Chubut, Argentina. 3 y 4 de Mayo de 2007. Publicado en las correspondientes memorias (ISBN 978-950-763-075-0). Págs 124-129.

Metaheurísticas basadas en Inteligencia Computacional Aplicadas a la Resolución de Problemas de Optimización Restringidos (Victoria Aragón, Leticia Cagnina y Susana Esquivel), en *VIII Workshop de Investigadores en Ciencias de la Computación (WICC 2006)*, en la Universidad Nacional de Morón, Buenos Aires, Argentina. 1 y 2 de Junio de 2006. Publicado en las correspondientes memorias (ISBN 950-9474-34-7). Págs 195-201.

A Particle Swarm Optimizer for Multi-Objective Optimization (Leticia Cagnina, Susana Esquivel y Carlos A. Coello Coello), en *XI Congreso Argentino en Ciencias de la Computación (CACIC 2005)*, Concordia, Entre Ríos. 17 al 21 de Octubre de 2005. Publicado en las correspondientes memorias (ISBN 950-698-166-3). Págs 54-65.

Optimización Mono y Multiobjetivo a través de Métodos de Aproximación de Soluciones en Ambientes Estacionarios y no Estacionarios (Susana Esquivel, Victoria Aragón y Leticia Cagnina), en *VII Workshop de Investigadores en Ciencias de la Computación (WICC 2005)*, realizado en la ciudad de Río Cuarto, Córdoba en Mayo de 2005. Publicado en las correspondientes memorias (ISBN 950-669-337-2). Págs 291-295.

Capítulo 2

Problemas de Optimización y Métodos de Resolución

La optimización es una importante tarea en distintas disciplinas tales como investigación operativa, matemática, economía, ingeniería, etc. y es de amplia aplicación para resolver problemas del mundo real de distinta naturaleza. En este capítulo se describen conceptos fundamentales de la optimización en general y se plantean algunos métodos de resolución comúnmente utilizados, con especial énfasis en los métodos heurísticos.

Contenido del Capítulo

2.1. El proceso de optimización

Una tarea de optimización es un proceso que consiste en encontrar la mejor solución candidata de entre una colección de alternativas. Esta tarea se plantea como un problema estructurado con funciones de variables de decisión, que deben o no satisfacer un conjunto de restricciones.
Un *problema de optimización* se conforma por una o varias funciones objetivo y (posiblemente) una o varias restricciones. Matemáticamente, este problema se describe mediante los siguientes elementos:

- **Variables de decisión**: contienen los valores que se modifican para resolver el problema.
- **Función/es objetivo**: pueden ser una o varias. Estas funciones se expresan en términos de las variables de decisión, y el resultado de su evaluación es el que se desea optimizar (maximizar o minimizar). Si sólo una función es considerada se habla de un problema de *optimización mono-objetivo*. Si varias funciones son consideradas, el problema se denomina optimización *multiobjetivo*.
- **Restricciones**: expresadas en forma de ecuaciones de igualdad o desigualdad, se deben cumplir o satisfacer para que la solución sea considerada factible, es decir, válida. Si el problema no presenta restricciones, todas las soluciones son válidas.

La presencia o no de restricciones en la definición del problema lo clasifica como *optimización con restricciones* o *sin restricciones*.
El problema de optimización puede ser expresado como uno de *programación no lineal* (NLP por sus siglas en inglés) cuyo objetivo es encontrar el vector $\vec{x}$, que optimiza la función:

$$f(\vec{x}) \text{ con } \vec{x} = (x_1, x_2, \ldots, x_D) \in \mathcal{F} \subseteq \mathcal{S} \subseteq \mathbb{R}^D \tag{2.1}$$

donde $\vec{x}$ está sujeto a:

$$g_i(\vec{x}) \leq 0 \quad i = 1, 2, \ldots, n \tag{2.2}$$

$$h_j(\vec{x}) = 0 \quad j = 1, 2, \ldots, p \tag{2.3}$$

$x_d \in [l_d, u_d]$ con $d \in [1, D]$, siendo l_d y u_d los límites inferior y superior en el que cada variable está definida. D es el número de variables del problema. Las funciones g_i y h_j están definidas en $\mathcal{S}$ (espacio de búsqueda) y son las funciones de restricciones de desigualdad e igualdad respectivamente (lineales o no). Toda restricción de desigualdad que satisfaga $g_j(\vec{x}) = 0$ se clasifica como "activa" en $\vec{x}$; de otro modo, estará inactiva. Toda restricción $h_j(\vec{x})$ es considerada activa en todos los puntos de $\mathcal{F}$, independientemente del valor de $\vec{x}$. El conjunto de soluciones $\mathcal{F}$ se define como la *región factible*.
Los NLP se caracterizan porque tanto la función objetivo como las restricciones pueden ser no lineales (los problemas de programación lineal sólo pueden incluir ecuaciones lineales).

A continuación se listan algunas definiciones útiles considerando problemas de optimización que se minimizan (sin pérdida de generalidad ya que cualquier problema de maximización se puede transformar en uno de minimización). El *óptimo* $\vec{x^*} = (x_1, x_2, \ldots, x_D)$ es un punto del espacio de búsqueda que corresponde al mejor valor posible de f en todo el espacio de búsqueda.

Definición 1: Mínimo global. *El mínimo global de una función $f(\vec{x})$ definida en un conjunto $\mathcal{S}$ es el punto $\vec{x^*} \in \mathcal{S}$ si y sólo si $f(\vec{x^*}) \leq f(\vec{x})$ para todo $\vec{x} \in \mathcal{S}$.*

Definición 2: Mínimo local. *El mínimo local de una función $f(\vec{x})$ definida en un conjunto $\mathcal{S}$ es el punto $\vec{y} \in \mathcal{S}$ si y sólo si $f(\vec{y}) \leq f(\vec{x})$ para todo $\vec{x} \in \mathcal{S}$ que se encuentre a una distancia $\epsilon > 0$ de $\vec{y}$.*

Definición 3: Función monótona. *Una función $f(\vec{x})$ es monótona creciente si para dos puntos cualesquiera $\vec{x_1}$ y $\vec{x_2}$ con $\vec{x_1} \leq \vec{x_2}$ se cumple que $f(\vec{x_1}) \leq f(\vec{x_2})$ y es monótona decreciente si $f(\vec{x_1}) \geq f(\vec{x_2})$.*

Definición 4: Función unimodal. *Una función $f(\vec{x})$ es unimodal en el intervalo $\vec{a} \leq \vec{x} \leq \vec{b}$ si y sólo si es monótona en cada lado del único punto óptimo $\vec{x^*}$ en el intervalo antes establecido. O sea, $\vec{x^*}$ es el único punto mínimo de f en el rango $[\vec{a}, \vec{b}]$ con lo cual $f(\vec{x})$ es unimodal en el intervalo si y sólo si para dos puntos cualesquiera $\vec{x_1} \leq \vec{x_2}$: $\vec{x^*} \leq \vec{x_1} \leq \vec{x_2}$ implica que $f(\vec{x^*}) \leq f(\vec{x_1}) \leq f(\vec{x_2})$ y $\vec{x^*} \geq \vec{x_1} \geq \vec{x_2}$ implica que $f(\vec{x^*}) \leq f(\vec{x_1}) \leq f(\vec{x_2})$.*

Definición 5: Función multimodal. *Una función $f(\vec{x})$ es multimodal si dados k intervalos distintos, cumple con la definición de ser unimodal en cada uno de los k intervalos, es decir, posee varios puntos óptimos.*

Definición 6: Función separable. *Una función $f(\vec{x})$ es separable si puede ser expresada como la suma de funciones objetivo parciales en las cuales una o varias variables de decisión pueden aparecer en cada una de ellas.*

Definición 7: Función convexa. *Una función $f(\vec{x})$ es convexa en $\mathbb{R}$ si para dos vectores cualesquiera $\vec{x_1} \leq \vec{x_2} \in \mathbb{R}$ se cumple $f(\theta\vec{x_1} + (1-\theta)\vec{x_2}) \leq \theta f(\vec{x_1}) + (1-\theta) f(\vec{x_2})$ donde θ es un escalar en el rango $0 \leq \theta \leq 1$. La función es estrictamente convexa si para $\vec{x_1} \neq \vec{x_2}$ el signo $\leq$ se reemplaza por $<$.*

Definición 8: Función cóncava. *Una función $f(\vec{x})$ es cóncava si cumple con la definición de convexa pero con las desigualdades revertidas, es decir, $f(\vec{x})$ es cóncava si $-f(\vec{x})$ es convexa.*

La figura 2.1 ilustra algunas de las definiciones anteriores.

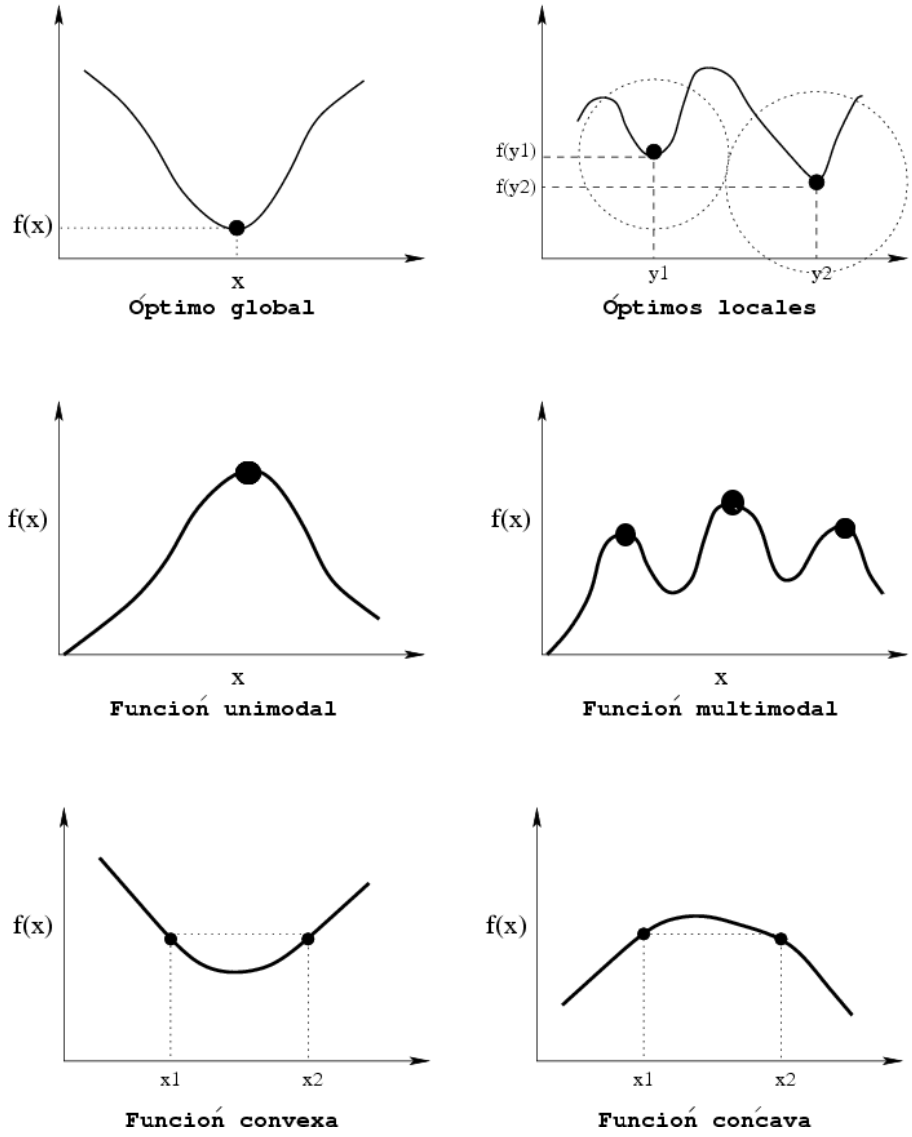


Figura 2.1: Tipos de funciones.

2.2. Optimalidad en problemas con restricciones

El mayor problema de las funciones con restricciones es el tratamiento de estas últimas. Por ello es preciso diferenciar dos casos posibles: las restricciones pueden tener efecto en el punto óptimo o no. Si las restricciones no influenciaran el hallazgo del óptimo, el proceso de optimización podría llevarse a cabo con cualquier método que maneje problemas sin restricciones. En la figura 2.2 puede observarse una función $f_1(\vec{x})$ con dos restricciones $g_1(\vec{x})$ y $g_2(\vec{x})$ que pueden ser excluidas (problema sin restricciones) porque no influyen en el valor óptimo.

El caso antes presentado no ocurre en la mayor parte de las funciones y, de hecho en la práctica, resulta muy difícil diferenciar si las restricciones influyen o no en el óptimo. Por ello, para poder resolver NLP generales, se asume que las restricciones ejercen influencia

sobre el óptimo, por lo que deben ser tenidas en cuenta. En la figura 2.3 puede observarse cómo el óptimo de $f_2(\vec{x})$ se encuentra en el borde de la zona factible delimitada por tres restricciones $g_1(\vec{x})$, $g_2(\vec{x})$ y $g_3(\vec{x})$. Por las características de la función (las restricciones influyen en el óptimo) se puede diferenciar entre el valor encontrado optimizando $f_2(\vec{x})$ sin considerar las restricciones (x^b) y el óptimo obtenido considerando las tres restricciones (x^*). Nótese que x^b es menor a x^*, pero la primera, al no encontrarse en la zona factible, no es considerada una solución válida.

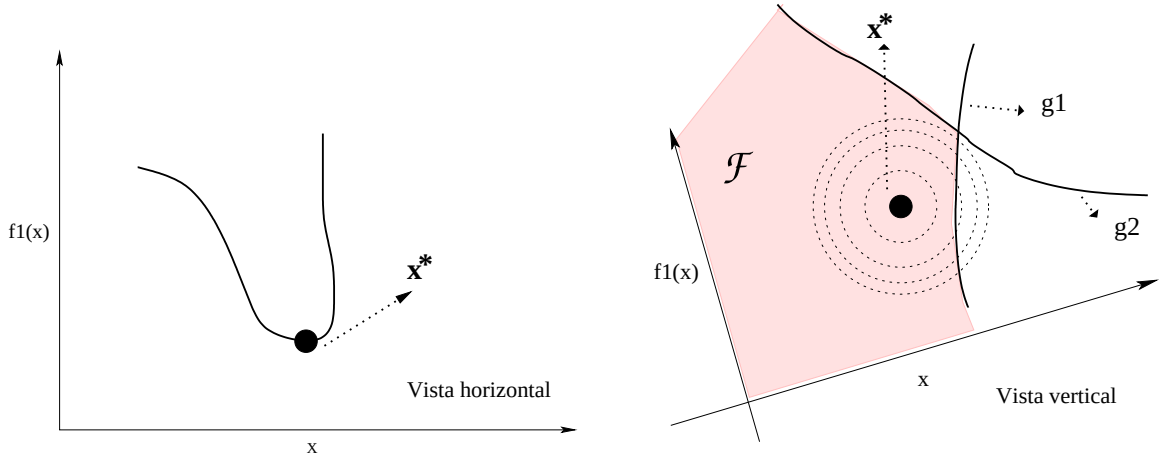


Figura 2.2: Las restricciones no tienen influencia en el óptimo de la función.

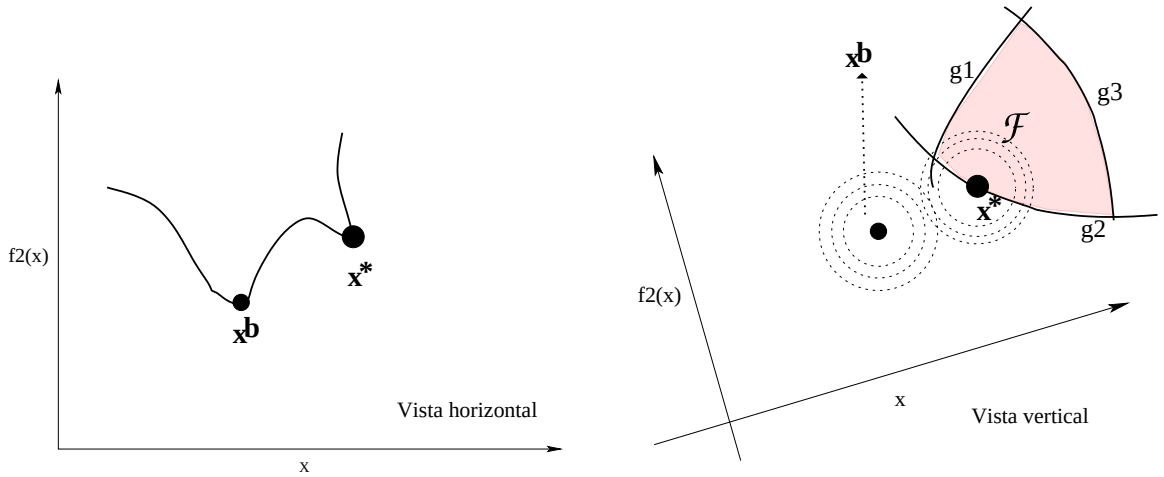


Figura 2.3: Las restricciones tienen influencia en el óptimo de la función.

Harold Kuhn y Albert Tucker presentaron ciertas condiciones de optimalidad necesarias y suficientes para encontrar una solución (el óptimo global) a un sistema de ecuaciones no lineal con $f(\vec{x})$, $g_i(\vec{x})$ y $h_j(\vec{x})$ diferenciables. Al problema de hallar ese valor óptimo se lo conoce como *problema de Kuhn-Tucker* y se define como:

Encontrar los vectores $\vec{x}$, $\vec{u}$ y $\vec{v}$ que satisfagan:

$$\nabla f(\vec{x}) - \sum_{i=1}^{I} u_i \nabla g_i(\vec{x}) - \sum_{j=1}^{J} v_j \nabla h_j(\vec{x}) = 0$$
$$g_i(\vec{x}) \geq 0 \quad \text{para} \quad i = 1, 2, \ldots, I$$
$$h_j(\vec{x}) = 0 \quad \text{para} \quad j = 1, 2, \ldots, J$$
$$u_i g_i(\vec{x}) = 0 \quad \text{para} \quad i = 1, 2, \ldots, I$$
$$u_i \geq 0 \quad \text{para} \quad i = 1, 2, \ldots, I$$

Los teoremas que se listan a continuación aseguran la existencia del valor óptimo.

Teorema 1: Necesidad de Kuhn-Tucker [154]. *Sean f, g y h funciones diferenciables y sea $\vec{x^*}$ una solución factible para el problema de Kuhn-Tucker. Sea $I = \{i | g_i(\vec{x^*}) = 0\}$. Además $\nabla g_i(\vec{x^*})$ para $i \in I$ y $\nabla h_j(\vec{x^*})$ para $j = 1, \cdots, p$ son linealmente independiente. Si $\vec{x^*}$ es una solución óptima al problema de Kuhn-Tucker, entonces existe un par $(\vec{u^*}, \vec{v^*})$, tal que $(\vec{x^*}, \vec{u^*}, \vec{v^*})$ resuelve el problema de Kuhn-Tucker.*

Al hecho de que se cumplan las condiciones listadas en el Teorema 1, o sea, que $\nabla g_i(\vec{x^*})$ y $\nabla h_j(\vec{x^*})$ sean linealmente independientes con el valor óptimo, se lo denomina *calificación de restricción*. La calificación de restricción implica ciertas condiciones de regularidad en la región factible que habitualmente suelen satisfacerse en problemas prácticos. El problema asociado a este hecho es que en la práctica es difícil verificar la calificación de una restricción ya que esto implica que se conozca el óptimo de antemano. No obstante, para algunos problemas particulares de programación no lineal en los que:

- todas las restricciones de desigualdad e igualdad son lineales,
- las restricciones de desigualdad son funciones cóncavas, las restricciones de igualdad son lineales y existe al menos una solución $\vec{x}$ factible que se encuentra estrictamente en el interior de la zona factible definida por las restricciones de desigualdad,

la calificación de una restricción siempre se satisface.

Cuando la calificación de restricción no se cumple en el óptimo, no es posible asegurar la existencia de una solución al problema de Kuhn Tucker (o sea que satisfaga las condiciones de Kuhn Tucker). Si la calificación de restricción se cumple en un punto factible cualquiera, es posible utilizar el Teorema 1 para probar que ese no es óptimo si no satisface las condiciones de Kuhn Tucker. Pero no todo punto que satisfaga las condiciones de Kuhn Tucker (punto de Kuhn Tucker) es un valor óptimo ya que ciertas condiciones deben establecerse (Teorema 2).

Teorema 2: Suficiencia de Kuhn-Tucker [154]. *Considerar el problema general de optimización no lineal. Suponer que la función objetivo $f(\vec{x})$ es convexa, las restricciones de desigualdad $g_i(\vec{x})$ son todas funciones cóncavas para $i = 1, \ldots, I$ y las restricciones de*

igualdad $h_j(\vec{x})$ para $j = 1, \ldots, J$ son lineales. Si existe una solución $(\vec{x^*}, \vec{u^*}, \vec{v^*})$ que satisface las condiciones del problema de Kuhn-Tucker, entonces $\vec{x^*}$ es una solución óptima al problema de optimización no lineal.

El Teorema 2 resume que si se cumplen esas condiciones, la obtención de un punto Kuhn Tucker equivale a obtener una solución óptima al problema de programación lineal. El teorema también puede utilizarse para demostrar si una solución a un problema de optimización no lineal es óptima.

La calificación de restricciones normalmente se cumplirá en muchos problemas prácticos con lo cual la determinación de puntos óptimos puede facilitarse. Si la función del problema a resolverse es diferenciable entonces un punto Kuhn Tucker es un candidato posible para ser óptimo. Usando esta afirmación es que muchos métodos de optimización no lineal intentan converger a un punto Kuhn Tucker. Además, cuando se cumplen las condiciones del Teorema de Suficiencia, un punto Kuhn Tucker automáticamente se convierte en un óptimo global. Pero no todo es tan simple y directo ya que las condiciones de suficiencia son difíciles de verificar y frecuentemente los problemas prácticos no poseen esas propiedades (una restricción de igualdad no lineal bastaría para violar las premisas del Teorema 2).

Tomando como base la dificultad en determinar la existencia de un punto óptimo es que existen numerosos métodos para resolver los problemas de optimización no lineales. La próxima sección describe brevemente algunos de ellos.

2.3. Métodos de resolución de problemas

A continuación se presentan diferentes tipos de métodos para resolver problemas en general. Nótese que no se requiere que el problema a tratar tenga determinadas características como que la función a optimizarse sea diferenciable, unimodal, mono-objetivo, etc. Si se consideran las características del problema, debe plantearse una clasificación diferente.

2.3.1. Métodos exactos

Estos métodos realizan una búsqueda exhaustiva de soluciones. A continuación se describen 3 de los métodos exactos más conocidos utilizados en la resolución de problemas de optimización.

Búsqueda Exhaustiva por Enumeración
Esta técnica, a pesar de ser una de las más simples, sólo es factible para aquellos problemas en los que el espacio de búsqueda es lo suficientemente pequeño y enumerable. En tal caso, se generan una a una todas las soluciones posibles y se elige aquella con mejor valor objetivo como la solución óptima del problema. Obviamente, para problemas con gran cantidad de soluciones, no es factible su utilización debido al empleo exagerado de

recursos (tiempo y espacio) requeridos. Algunas características que posee este método son [123]:

- Todas las soluciones deben ser generadas para poder elegir la mejor.
- No importa el orden en que se generen las soluciones, ni el orden en que se evalúan (¡todas se evaluarán!).
- Generalmente son algoritmos simples de programar: lo único que se requiere es la generación sistemática de cada posible solución del problema, evitando repeticiones.
- Pueden incorporarse técnicas para reducir la cantidad de tiempo empleado (*backtracking*).

Divide y Vencerás

Mediante este método, se resuelve el problema recursivamente a partir de la solución de subproblemas del mismo tipo, pero de menor tamaño. Si los subproblemas continúan siendo relativamente grandes, la técnica es aplicada nuevamente hasta descomponerlos en subproblemas suficientemente pequeños como para resolverlos directamente. La resolución de un problema mediante este método consta fundamentalmente de los siguiente pasos:

1. *Fase de División.* Plantear el problema de tal manera que pueda ser descompuesto en k subproblemas del mismo tipo, pero de menor tamaño.
2. Resolver independientemente todos los subproblemas directamente, si es que son elementales, o en forma recursiva, en caso de no serlo. Requiriendo que el tamaño de los subproblemas sea menor que el del problema original, se garantiza la convergencia de los mismos a los casos elementales, también denominados casos *base*.
3. *Combinar* las soluciones obtenidas en el paso anterior para construir la solución del problema original.

Como características importantes se pueden mencionar [73]:

- El método constituye una potente herramienta para solucionar problemas complejos.
- Para poder aplicarlo debe ser posible la división del problema en subproblemas de menor tamaño.
- El algoritmo que lo implementa suele ser eficiente.
- La adaptación del algoritmo a entornos multiprocesador hace factible la rápida ejecución del método.
- Se realiza un uso eficiente de las memorias *caché* (mucho más rápidas que la memoria principal) debido a que al dividirse el problema en otros de menor tamaño, estos últimos pueden ser resueltos en las memorias rápidas sin hacer uso de la memoria principal (de acceso lento).

- El proceso de resolución puede tornarse lento debido al gasto indirecto de recursos por las llamadas recursivas en la fase de división.

Ramificación y Acotación
Este método de exploración dirigida, conocido también como *Branch and Bound*, consiste de un número de etapas consecutivas mediante las cuales se detecta la solución en un proceso ordenado de exploración.
Inicialmente el espacio de soluciones es particionado en subconjuntos siguiendo algún criterio. Cada uno de estos subconjuntos está representado por un nodo (punto de decisión) en un árbol y cada nodo tiene asignado un valor de cota inferior para la función objetivo. Este valor de cota se emplea para descartar aquellos nodos que no son prometedores, y así explorar los de mejor valor de cota por medio de nuevas ramificaciones. Los nodos del último nivel del árbol (hojas) que no pueden ramificarse, representan soluciones completas al problema y la menor de ellas (si se está minimizando) es la óptima.

2.3.2. Métodos de exploración dirigida

Este tipo de métodos se caracteriza porque las soluciones son encontradas iterativamente guiando la búsqueda con algún criterio. Dos de los más utilizados son los que se describen a continuación.

Programación Dinámica
Existen algunos problemas en los que no es factible la división en subproblemas independientes. Cuando los subproblemas obtenidos no son independientes sino que existe solapamiento entre ellos, una solución recursiva no resulta eficiente debido a la repetición de cálculos que se producirá en la resolución de cada uno de los subproblemas. En estos casos la programación dinámica ofrece una solución aceptable resolviendo los subproblemas solamente una vez y guardando esas soluciones en una tabla para su futura utilización. La solución de problemas mediante esta técnica se basa en el *principio de optimalidad* enunciado por Bellman en 1957 que dice: "En una secuencia de decisiones óptima toda subsecuencia ha de ser también óptima" (principio no siempre aplicable a todo problema). Para que un problema pueda ser abordado por esta técnica ha de cumplir dos condiciones [63]:

- La solución al problema debe ser alcanzada a través de una secuencia de decisiones, una en cada etapa.

- Dicha secuencia de decisiones ha de cumplir el principio de optimalidad.

Los pasos que realiza el método son:

1. Planteamiento de la solución como una sucesión de decisiones y verificación de que cumple el principio de optimalidad.

2. Definición recursiva de la solución.

3. Cálculo de la solución óptima haciendo uso de la información obtenida.

Esta técnica presenta una limitación importante: no todos los problemas pueden ser descompuestos en subproblemas que se pueden resolver independientemente.

Algoritmos Voraces o Greedy

Estos algoritmos toman decisiones basándose en la información que tienen disponible en forma inmediata sin tener en cuenta los efectos que estas decisiones puedan tener en el futuro. El algoritmo selecciona la tarea que parezca más prometedora en un determinado instante; nunca reconsidera su decisión, sea cual fuere la situación que pudiere surgir más adelante. No hay necesidad de evaluar alternativas, ni de emplear sofisticados procedimientos de seguimiento que permitan deshacer las decisiones anteriores. Estos algoritmos están caracterizados por una *función solución* que comprueba si un cierto número de candidatos constituye una solución al problema, ignorando si es o no óptima; una *función de factibilidad* que comprueba si un cierto conjunto de candidatos es factible como solución al problema; una *función de selección* que indica cuál de los candidatos a solución es el más prometedor (que aún no ha sido rechazado ni seleccionado), y una *función objetivo* que evalúa la solución encontrada y retorna el valor de la misma. Estos algoritmos trabajan inicialmente con un conjunto vacío de elementos y paso a paso avanzan hasta encontrar una buena solución.
Las características que los identifican son [183]:

- Escogen el mejor candidato para formar parte de la solución.
- Esa selección es única e inmodificable. No deshacen una selección ya realizada: una vez incorporado un elemento a la solución permanece hasta el final, lo mismo sucede si es un candidato rechazado.
- La calidad de estos algoritmos está en relación con las características de las instancias que pretenden resolver: obtiene buenos resultados para algunas instancias fáciles y malos para otras.
- Caen fácilmente en óptimos locales debido a que no analizan los vecindarios de las soluciones (criterio voraz).

2.3.3. Métodos heurísticos

Es importante destacar que cuando el número de variables del problema es muy elevado, los métodos exactos (y en muchos casos los de exploración dirigida también) dejan de ser eficientes a causa de la gran cantidad de tiempo y recursos que insumen para encontrar una solución óptima. Esta ineficiencia dio origen a una clase de procedimientos simples que tratan de encontrar, de modo eficiente, una solución factible cercana a la óptima. Estos procedimientos reciben el nombre de *heurísticas* (del griego *heuriskein*: encontrar o descubrir).
Formalmente, una heurística es una técnica que trata de encontrar soluciones buenas (óptimas o casi óptimas) con un costo computacional razonable sin "garantizar" su factibilidad ni su optimalidad. De hecho, en muchos casos tampoco se puede establecer qué tan

cerca a la optimalidad se encuentra la solución hallada [47].
En contraposición a los métodos exactos o a los de exploración dirigida que proporcionan una solución óptima, las heurísticas se limitan a proporcionar buenas soluciones aunque no necesariamente óptimas. Algunas razones por las cuales se utilizan métodos heurísticos son:

- No se conoce ningún método exacto que resuelva el problema.
- El método exacto es muy costoso, computacionalmente hablando.
- El método heurístico es más flexible que el exacto permitiendo la incorporación de condiciones o restricciones difíciles de modelar matemáticamente.
- El método heurístico se puede utilizar como parte de un procedimiento global que garantice encontrar el óptimo del problema.

Abordando el estudio de algoritmos heurísticos es posible comprobar que dependen en gran medida del problema concreto para el cual se han diseñado, a diferencia de algunos métodos de resolución de propósito general como los exactos o de exploración dirigida en los cuales existe un procedimiento conciso y preestablecido independiente, en gran medida, del problema abordado. Por ello, se describen en este capítulo algunas de las heurísticas más utilizadas en el contexto de optimización, siendo el objetivo proporcionar una visión general de cada uno de estos métodos aunque sin profundizar en su estudio ya que cada uno de ellos podría requerir de varias páginas para su descripción detallada.

I. **Métodos heurísticos de Búsqueda Local**
   Estas técnicas basan su funcionamiento en el refinamiento iterativo de una solución factible realizando pequeñas perturbaciones en su estructura combinatoria. Los algoritmos de esta clase inicialmente construyen una solución inicial (al azar o con el uso de alguna otra heurística), y luego iterativamente se buscan nuevas soluciones que mejoren las actuales. A continuación se listan algunas de las más empleadas.

   Recocido Simulado (*Simulated Annealing*)
   Este método se basa en conceptos de física que describen el proceso sufrido por los cuerpos sólidos o líquidos al ser sometidos a un baño térmico. Al decrementarse la temperatura del cuerpo, la movilidad de las moléculas también lo hacen, alineándose en una estructura cristalina. Esta estructura representa un estado de energía (el mínimo posible) en el sistema. Para asegurar que la estructura cristalina sea alcanzada, el procedimiento de enfriado debe realizarse tan lento como sea posible, ya que de no ser así se podría obtener una estructura amorfa.
   En el contexto de optimización, el valor mínimo esperado se representa con el mínimo estado de energía del sistema [86]. Este método utiliza un procedimiento de ascenso (o descenso) de gradiente, que escoge una solución (aunque no siempre la mejor). Dependiendo de una *temperatura*, la probabilidad de seleccionar una solución peor que la actual va descendiendo a medida que transcurre el tiempo. La temperatura se actualiza iterativamente (*procedimiento de enfriado*) hasta que, al

final del algoritmo, cuando la temperatura es cero, se escoge de forma determinística siempre la mejor solución. La actualización de la temperatura debe ser realizada cuidadosamente ya que si se acelera demasiado, el algoritmo quedaría atrapado en un óptimo local. En cambio, si se realiza muy lentamente, el proceso de búsqueda de la solución óptima insumiría más tiempo del necesario.
Es fundamental en este algoritmo elegir una buena temperatura inicial y un correcto procedimiento de enfriado.

Búsqueda Tabú
Esta técnica consiste en un algoritmo de búsqueda de vecindario iterativo [67], que emplea el concepto de memoria y lo implementa con estructuras simples, con el objetivo de dirigir la búsqueda de la solución final en función de los resultados ya obtenidos. La técnica utiliza una memoria (denominada *lista tabú*) que almacena los últimos movimientos realizados, de forma tal que no se caiga en los óptimos locales que allí almacena. De esta manera, el algoritmo hace uso de su historia pasada para explorar nuevas áreas. En cada iteración se elige la dirección a tomar de entre varias posibles. Luego, haciendo uso de la información guardada en la lista tabú, se eliminan aquellas direcciones que no deberían explorarse nuevamente. Además de la memoria a corto plazo, diversas variantes incluyen una memoria a mediano plazo que registra los atributos más comunes de un conjunto de soluciones de forma tal que se explore esa zona del espacio de soluciones, y una memoria a largo plazo, que diversifica la búsqueda sobre regiones no exploradas aún. Algunas características que posee este método de búsqueda son [123]:

- La búsqueda que se realiza es básicamente determinística (en oposición al Recocido Simulado), aunque pueden agregársele algunos elementos probabilísticos [58].
- El método fuerza a explorar nuevas áreas del espacio de búsqueda.
- La técnica fue diseñada originalmente para escapar de óptimos locales.
- Se realizan movimientos ascendentes (en inglés *uphill*) como los que se efectúan en el Recocido Simulado, sólo cuando se ha alcanzado un óptimo local.
- Se trabaja con soluciones completas lo cual implica que si el algoritmo es finalizado antes de tiempo puede retornarse, una solución completa al problema.
- El algoritmo que implementa el método emplea varios parámetros, con lo cual un nuevo desafío es la selección correcta de los mismos.

Ascenso de Colina (*Hill Climbing*)
Estos algoritmos son típicamente locales ya que deciden qué hacer teniendo en cuenta las consecuencias inmediatas de sus opciones. Muchos algoritmos heurísticos se basan en este método. Una de las tareas más importantes que llevan a cabo consiste en escoger el tamaño del paso correcto para ascender, además de iniciar correctamente el "escalado" del espacio de búsqueda. Una de las desventajas que posee esta técnica es que si la búsqueda se queda atrapada en estados lejanos al objetivo y

desde los cuales no se puede salir, nunca podrá encontrar una solución al problema. Algunos de los estados que se pueden alcanzar son:

- Un óptimo local, que es mejor que sus estados vecinos, aunque no mejor a otros más alejados.
- Una meseta, espacio de búsqueda en el que todo conjunto de estados vecinos tiene igual valor objetivo.
- Un risco, tipo especial de óptimo local, imposible de atravesar con movimientos simples.

Existen algunos caminos a seguir frente a este tipo de situaciones, aunque no se garantiza su solución:

- Para evitar los óptimos locales se puede regresar a un estado anterior y explorar una dirección diferente (*backtracking*).
- Para las mesetas se puede dar un salto grande en alguna dirección para tratar de explorar una nueva sección del espacio de búsqueda.
- Para los riscos, aplicar una o más reglas antes de realizar una prueba del nuevo estado, como moverse en varias direcciones a la vez.

GRASP (*Greedy Randomized Adaptive Search Procedure*)
GRASP es un método miope aleatorizado y adaptativo que se basa en la combinación de fases de exploración. En éstas se utilizan algoritmos de búsqueda local con procedimientos aleatorios para expandir el espacio de búsqueda recorrido. Entre las diferentes fases se identifican [141]:

a) Fase constructiva: se escoge iterativamente y al azar un elemento de una lista restringida de candidatos.

b) Fase de postprocesamiento: se mejora la solución obtenida en el paso anterior. Para este paso suele emplearse una búsqueda local simple descendente.

Los principales elementos que componen y determinan la técnica GRASP son: la función heurística, la forma en la que se construye la lista restringida de candidatos, el método de postprocesamiento y el criterio de parada.

Algunas características se pueden resumir en [141]:

- El algoritmo es iterativo y constructivo (se va agregando un elemento por iteración).
- La función objetivo es importante en la fase constructiva de la solución porque al finalizar las iteraciones se espera una solución de alta calidad.
- La función miope es la que guía la búsqueda en la construcción de las soluciones, de forma tal que se efectúe el mejor movimiento en cada paso.
- Utiliza una medida miope adaptativa para tomar en cuenta las decisiones previamente tomadas.

- En cada etapa se construye una lista de candidatos admisibles ordenada de manera decreciente con respecto a su beneficio medido por la función miope.

II. **Computación Evolutiva**

Esta categoría de métodos heurísticos tiene fundamento en la selección natural que sufren los individuos y se aplica a un ambiente artificial como es el de computación. La selección natural puede ser vista como un proceso de optimización en el cual los individuos de la población van mejorando su calidad para adaptarse al medio donde subsisten. En el ámbito de la computación evolutiva, este hecho también se observa haciendo referencia a un *individuo* como una solución potencial a un problema específico, el cual interactúa en un medio compuesto por la función objetivo y las restricciones. La tarea es decidir cuán apto es ese individuo para sobrevivir en ese medio.

La computación evolutiva involucra métodos poblacionales, es decir, métodos que trabajan con varias soluciones a la vez y no solamente con una como lo hacen muchas heurísticas.

Un algoritmo evolutivo consta de una población inicial de posibles soluciones (individuos) a la cual se le aplican operadores probabilísticos para obtener nuevas soluciones, las cuales se conservan o descartan mediante un mecanismo de selección. Este proceso iterativo recibe el nombre de *generaciones*, las cuales dependen del usuario o está definido por el propio algoritmo. Los operadores probabilísticos más comunes son la *recombinación*, que realiza una mezcla de dos o más individuos llamados *padres* para obtener uno o más individuos nuevos llamados *hijos*; y la *mutación*, que consiste en una alteración aleatoria sobre un individuo para cambiarlo.

La computación evolutiva puede ser dividida en diferentes paradigmas [93], los cuales comparten todas o algunas de las características antes mencionadas. Cada uno de ellos: *Programación Evolutiva, Estrategias Evolutivas, Algoritmos Genéticos* y *Programación Genética*, se describen brevemente a continuación.

Programación Evolutiva

Fue propuesta por Lawrence J. Fogel [51], como una técnica de búsqueda en un espacio de pequeñas máquinas de estados finitos. Posteriormente fue utilizada como optimizador. El proceso de evolución consiste en encontrar un conjunto de comportamientos adecuados de un espacio de comportamientos observados. La función de aptitud (función objetivo) mide el "error en el comportamiento" de un individuo con respecto al ambiente al cual pertenece.

La programación evolutiva posee las siguientes características [162]:

- Utiliza una representación para los individuos adaptada al problema concreto que trata de resolver.
- La mutación se realiza de acuerdo a la representación utilizada y es el único operador que utiliza para producir nuevos hijos. Generalmente se utiliza

un *Torneo Probabilístico* para seleccionar al mejor individuo que pasará a la próxima generación (*estrategia elitista*).

- Si bien no se realiza cruce de individuos, la mutación es lo suficientemente flexible como para producir un efecto similar al del cruce.

Estrategias Evolutivas
Estas estrategias fueron propuestas originalmente por Ingo Rechenberg [153], utilizando selección, mutación y población de tamaño uno. Posteriormente, Hans-Paul Schwefel [167] introdujo el cruce y poblaciones con más de un individuo. Están basadas en el concepto de la *evolución de la evolución*. La idea es optimizar el proceso evolutivo utilizando estrategias que lo influencian por medio de su adaptación en paralelo con la población.

Algunas características son:

- La representación para los individuos se basa en la información genética propia del individuo. Cada individuo tiene, además, un conjunto asociado de parámetros estratégicos.
- La mutación se realiza tanto en el material genético como en los parámetros del algoritmo.
- Al igual que la mutación, el cruce de individuos involucra la recombinación de la carga genética y de las estrategias de los individuos seleccionados como padres.
- Luego de evaluar los k hijos, se seleccionan n descendientes de forma determinística mediante algún método. Por ejemplo: (n, k) selecciona los mejores n hijos, o $(n + k)$ selecciona los mejores n hijos y padres (elitista).

Al igual que en la programación evolutiva se realiza la mutación adaptativa, pero a diferencia de la otra, el cruce sí tiene un papel importante, especialmente para adaptar la mutación.

Algoritmos Genéticos
Estos algoritmos emulan la evolución genética. Las características de los individuos están expresadas con base en genotipos. Originalmente fueron presentados por John Holland en los 1960s [70]. Muchas variaciones han sufrido los algoritmos originales de Holland, para los cuales se han desarrollado una gran variedad de formas de representación, métodos de selección y operadores de recombinación, así como también de mutación y elitismo [6] [7].

La principales características de estos algoritmos son:

- La representación de los individuos se realiza a través de la utilización de cadenas binarias (a diferencia de los dos últimos paradigmas que no requieren codificación específica alguna).

- Utiliza selección proporcional: se elige el mejor individuo con base en una distribución de probabilidades proporcional a la función objetivo [61].
- El operador principal es la recombinación, mientras que la mutación es sólo un operador secundario.

Programación Genética

La programación genética es una especialización de los algoritmos genéticos [46], y al igual que ésta, se concentra en la evolución de los genotipos. Sin embargo, a diferencia de los algoritmos genéticos, los individuos se representan en este caso como árboles (estructuralmente). Esta idea fue desarrollada por John Koza [89] [90] quien la utilizó para evolucionar programas ejecutables. A partir de allí se emplearon para evolucionar árboles de decisión para aplicaciones de minería de datos [43] [53], entre muchas otras áreas. En este paradigma cada individuo representa un programa computacional el cual se codifica utilizando una estructura de árbol. Se utilizan gramáticas para definir el problema. La función de aptitud depende exclusivamente del problema que se esté tratando. Los operadores de cruza sobre las estructuras de árbol son implementados seleccionando aleatoriamente un nodo de cada uno de los árboles padre, y reemplazando el subárbol correspondiente en uno de los padres por el del otro. Los operadores de mutación eligen un nodo aleatoriamente y lo reemplazan por otro generado al azar.

III. **Algoritmos Bioinspirados**

Así como la observación de la naturaleza ha sido una de las principales fuentes de inspiración para la propuesta de nuevos paradigmas computacionales, también lo ha sido el comportamiento de los seres vivos. Así es que una nueva categoría de algoritmos la conforman aquellos que están inspirados en esta analogía: los algoritmos bioinspirados. Algunos de los más conocidos son los *Ant Colony* (colonias de hormigas), *Algoritmos Culturales*, *Sistema Inmune Artificial* y *Particle Swarm Optimization* (cúmulo de partículas).
A continuación se realiza un breve resumen de estos métodos.

Ant Colony Optimization

La optimización basada en colonias de hormigas estudia el comportamiento de estos insectos sociales que evidencian la colaboración entre sí para obtener comportamientos complejos. Una de las características más importantes de las hormigas es que son capaces de encontrar el camino más corto entre su hormiguero y la fuente de alimento. Esto es posible gracias a una sustancia llamada feromona, que depositan para poder seguir el rastro del camino encontrado. Este proceso se dice que es de "autorefuerzo", ya que cada hormiga que sigue el camino señalado, deposita nuevamente feromona haciendo más fuerte el rastro. Así mismo, cuando un camino no es el ideal, y por lo tanto las hormigas no lo siguen, se va evaporando la feromona en ausencia del autorefuerzo, haciéndolo menos propenso a ser elegido.
Los algoritmos que simulan esta analogía, son del tipo constructivo: en cada iteración, cada hormiga o individuo construye una solución al problema recorriendo

un grafo de construcción. Cada arista del grafo representa los posibles pasos que la hormiga puede elegir y tiene asociada dos tipos de información que guían el comportamiento del individuo [141]:

- **Información heurística**: mide la preferencia heurística de moverse desde el nodo r hasta el nodo s. Las hormigas no modifican esta información durante la ejecución del algoritmo.
- **Información de los rastros de feromona artificial**: mide la "deseabilidad aprendida" del movimiento de un lugar a otro. Con esto se imita a la feromona real que depositan las hormigas. Esta información sí se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas.

Algoritmos Culturales

Los algoritmos culturales están basados en la teoría de sociólogos y arqueólogos que han tratado de modelar la evolución cultural de los individuos. Tales investigaciones indican que la evolución cultural puede ser vista como un proceso de herencia en dos niveles distintos: el nivel micro-evolutivo, que consiste en el material genético heredado por los padres, y el nivel macro-evolutivo, que es el conocimiento adquirido por los individuos a través de las generaciones. Este conocimiento, una vez que ha sido codificado y almacenado, sirve para guiar el comportamiento de los individuos que pertenecen a la población. Robert Reynolds [157] intentó captar ese fenómeno de herencia doble, de forma tal que incremente las tasas de aprendizaje o convergencia, haciendo así que el sistema responda mejor a un gran número de problemas. Estos algoritmos operan en dos espacios. El espacio de *población* que, al igual que los métodos evolutivos, es el que tiene un conjunto de individuos, los cuales podrán ser reemplazados por otros, dependiendo de su aptitud. El otro espacio es el de *creencias* donde se almacenan los conocimientos adquiridos por los individuos de las generaciones anteriores.

Sistema Inmune Artificial

Bajo este nombre se conoce al sistema adaptativo inspirado en la teoría del sistema inmune que poseen los seres vertebrados. Tanto desde el punto de vista biológico como computacional este nuevo paradigma [33] de optimización comparte el aprendizaje y la información guardada, ya sea sobre infecciones (en el sistema inmune real) como sobre información relevante sobre el espacio de búsqueda explorado (sistema computacional) para sortear problemas en situaciones futuras. Exiten actualmente muchos modelos de sistemas inmunes que se aplican en la resolución de variados problemas. Dependiendo del modelo que se utilice es el algoritmo que se debe programar (no existe un algoritmo único) aunque los más destacados son los basados en el proceso de creación de defensores del sistema y en el reconocimiento de invasores. Independientemente del modelo seleccionado, Nunes De Castro y Timmis [131] sugieren utilizar un esquema algorítmico que incluya: la representación adecuada de los componentes del sistema, un conjunto de mecanismos capaces de evaluar las

interacciones individuo-ambiente e individuo-individuo y, finalmente, un proceso de adaptación que gobierne las dinámicas del sistema general. El sistema inmune artificial ha sido aplicado exitosamente en problemas de robótica, seguridad en redes, optimización y, reconocimiento de patrones.

Particle Swarm Optimization
Esta técnica, inspirada en los patrones de vuelo de aves, simula los movimientos de una bandada que intenta encontrar comida. En este tipo de algoritmo, el movimiento de cada individuo o *partícula* se ve afectado por el movimiento del mejor de toda la *población* (*swarm*) o del *vecindario* al que pertenezca. La trayectoria de las partículas está definida por la *velocidad* asociada a cada una de ellas, y es la que regula la exploración del espacio de búsqueda. Cada partícula recorre el espacio ayudada por dos valores adicionales: el mejor alcanzado por la partícula hasta el momento y el mejor alcanzado por alguna partícula de la población. De esta manera se efectúa una búsqueda "guiada" por los espacios más prometedores.
En el próximo capítulo se describirá con mayor detalle esta heurística, así como también su aplicación a la resolución de problemas particulares.

Todos los métodos de resolución de problemas parecerían ser promisorios de obtener buenos resultados para determinados tipos de problemas, pero ¿cuál es el mejor de todos? En la próxima sección se presenta una respuesta a esta intricada cuestión.

2.4. Los teoremas de *No Free Lunch*

Los teoremas de *No Free Lunch* [196] establecen que por cada par de algoritmos de búsqueda que se considere, hay tantos problemas en los que el primer algoritmo funciona mejor que el segundo, como problemas en los que el segundo algoritmo supera al primero. Es decir, no es posible establecer qué algoritmo funciona mejor en todas las clases de problemas posibles. De hecho, es incluso posible, que uno de los mejores algoritmos para determinada función encuentre una solución peor que una búsqueda aleatoria para otra función. La conclusión directa de los teoremas es que considerando todos los posibles problemas y todos los posibles algoritmos, en promedio todos se comportan de igual manera. Además de proveer un marco de investigación para la relación problema−solución, los teoremas de *No Free Lunch* proporcionan una interpretación geométrica sobre el significado de que un algoritmo sea eficaz para cierto tipo de problema. También exponen información teórica sobre el procedimiento de búsqueda, funciones que varían en el tiempo, medidas formales para la evaluación de los algoritmos y dificultad sobre la optimización de los problemas en general.

Capítulo 3

La Heurística Particle Swarm Optimization

Una heurística de Inteligencia Colectiva basada en la naturaleza dinámica y adaptativa de la vida surgió en el año 1995 como una eficaz técnica de optimización: Particle Swarm Optimization. En este capítulo se describen las principales componentes de la heurística, los modelos mono-objetivo más característicos y, finalmente, algunas aplicaciones exitosamente resueltas con Particle Swarm Optimization.

Contenido del Capítulo

3.1. El paradigma Swarm Intelligence

Swarm Intelligence o Inteligencia Colectiva (IC) es un paradigma que agrupa técnicas de inteligencia artificial basadas en el estudio del comportamiento colectivo observado en la naturaleza. Este comportamiento es intrínseco a los orígenes de los individuos, los cuales tienen una fuerte tendencia a asociarse a otros y socializar [84]. De esta forma, reglas, conocimiento, experiencias y creencias pueden intercambiarse.
La IC consta de una población de agentes simples interactuando localmente con los demás y con el ambiente que los rodea. En general no existe una estructura de control centralizada previendo cómo deberían actuar los agentes individuales. No obstante, las interacciones locales entre tales agentes a menudo convergen en un *comportamiento global* del cual todos se benefician. Ejemplos de esto son las colonias de hormigas, los cardúmenes, las bandadas y las manadas, en donde todos los agentes se mueven en conjunto.
Se han efectuado diversos estudios para entender el comportamiento coordinado de los grupos, y éstos han sido utilizados en diferentes heurísticas de inteligencia colectiva. Una de ellas es la que se describe a continuación.

3.2. La heurística Particle Swarm Optimization

En el año 1995, James Kennedy y Russell Eberhart propusieron una heurística llamada *Particle Swarm Optimization* (PSO). Esta heurística se basa en el modelo psico-social de entidades colectivas. PSO posee influencia y aprendizaje social [85] que se refleja en cada agente de la entidad. Los individuos (*partículas*) en la heurística emulan una característica simple: adaptan su comportamiento al de los individuos dentro de su propio vecindario o de la población completa (en inglés, *swarm*).
El algoritmo fue inspirado en los patrones de vuelo de algunas aves [39]; imita una bandada recorriendo un espacio, probando diferentes lugares y convergiendo la bandada o cúmulo (*swarm*) a las mejores regiones, es decir, aquéllas donde se encuentra el alimento, o la protección, o el objetivo que sea.
Una partícula se mueve dentro del espacio de búsqueda siguiendo una trayectoria definida por su velocidad, y la memoria de dos buenos valores: el mejor encontrado por ella misma en su pasado y el mejor valor encontrado por alguna partícula de la población. Estos dos valores generan una "atracción" hacia los lugares más prometedores, y a esta atracción o fuerza se la denomina *presión social*.
A continuación se describen los principales elementos que caracterizan a una partícula.

3.2.1. Estructura de una Partícula

La estructura básica de una partícula es levemente más complicada que la de un individuo de un algoritmo genético. Consta de cinco componentes:

- Un vector $\vec{x}$ que describe la *ubicación* de la partícula dentro del espacio de soluciones. El tamaño de este vector depende del número de variables necesarias para resolver el problema. Cada posición de este vector se denomina *dimensión*.

- **Valor objetivo o aptitud *fitness***, representa la calidad de la solución representada por el vector $\vec{x}$, obtenido a través del cálculo de la *función de evaluación* correspondiente al problema específico.

- Un vector $\vec{v}$ que representa la *velocidad* de la partícula. Este vector, al igual que $\vec{x}$, se modifica en cada iteración del algoritmo reflejando así el cambio de dirección que sufre la partícula dentro del espacio de búsqueda.

- ***pbest***, mejor valor de *fitness* encontrado por la partícula hasta el momento.

- ***gbest***, mejor valor de *fitness* encontrado por alguna partícula del *swarm*. Este valor está presente si se utiliza un modelo global, es decir, un modelo en el que los individuos son influenciados por el mejor de toda la población.

- ***lbest***, mejor valor de *fitness* encontrado en el *vecindario* de una partícula. Este valor está presente si se utiliza un modelo local, es decir, un modelo en el que los individuos son influenciados por el mejor de un grupo pequeño de individuos (vecindario). La determinación del vecindario depende de la forma en la que se efectúa el agrupamiento de individuos de la población.

3.2.2. Trayectoria de la Partícula

La trayectoria de la partícula dentro del espacio de búsqueda está definida con base en el vector de ubicación $\vec{x}$ y el de velocidades $\vec{v}$, los cuales son sumados para obtener la nueva dirección. Justamente estos cambios de trayectoria son los que determinan la característica principal del algoritmo PSO, ya que a través de los mismos las partículas son forzadas a buscar soluciones en las áreas más promisorias del espacio de búsqueda. Cabe destacar que si los valores de $\vec{v}$ no se modificaran, la partícula sólo se movería con pasos uniformes en una única dirección.
Más específicamente, en cada iteración del algoritmo, la dirección que tomará la partícula es modificada considerando el valor de ***pbest*** (la atracción hacia la mejor posición alcanzada por la partícula) y el vector ***gbest*** (la atracción hacia la mejor posición alcanzada por alguna partícula del cúmulo o ***lbest*** si se consideran vecindarios). Además PSO introduce valores aleatorios de ajuste para las dos últimas atracciones. Estos valores aseguran que el tamaño del paso que realizan las partículas dentro del espacio sea variable, asegurando que las mismas no "caigan en una rutina", es decir, que no se muevan siempre con el mismo paso. La figura 3.1 ilustra la obtención de la nueva posición (**x'**) de la partícula **x** como consecuencia de la suma de la velocidad y las memorias de los mejores.

La influencia relativa de la memoria que lleva asociada cada partícula (***pbest***) es denominada *influencia cognitiva*, mientras que la memoria del mejor valor encontrado por alguna partícula de la población (***gbest*** o ***lbest***) es denominada *influencia social*. Ambas influencias determinan la *velocidad de aprendizaje* del cúmulo y esto permite determinar con qué rapidez el cúmulo convergerá a la solución.
Matemáticamente, el proceso de aprendizaje puede describirse utilizando una fórmula

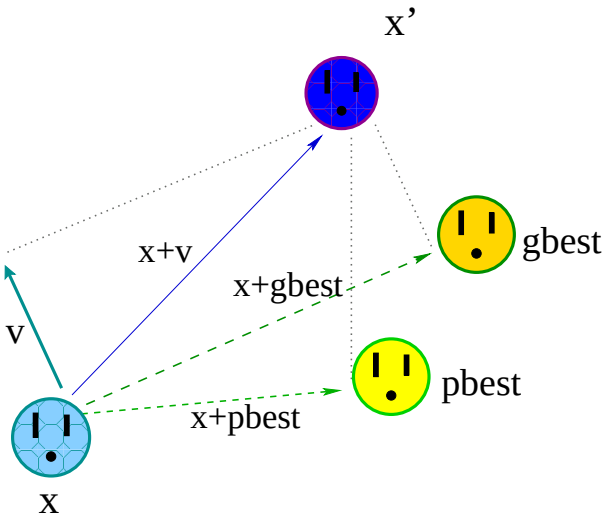


Figura 3.1: Trayectoria de la partícula x.

para la actualización de la velocidad de cada partícula y otra para la actualización de su posición dentro del espacio de búsqueda. Esto se describe en la siguiente subsección.

3.2.3. Algoritmo PSO básico

Considerando un problema de D-dimensiones, para cada partícula i de d dimensiones con $d \in [1, D]$, el aprendizaje está determinado por las siguientes ecuaciones:

$$v_{id} = v_{id} + \rho_1 * r_1 * (\boldsymbol{pbest}_{id} - x_{id}) + \rho_2 * r_2 * (\boldsymbol{gbest}_d - x_{id}) \tag{3.1}$$

$$x_{id} = x_{id} + v_{id} \tag{3.2}$$

donde v_{id} es el valor de la velocidad de la partícula i en la dimensión d, ρ_1 es el factor de aprendizaje cognitivo, ρ_2 es el factor de aprendizaje social, r_1 y r_2 son valores aleatorios uniformemente distribuidos en el rango $[0, 1]$, x_{id} es la posición corriente de la partícula i en la dimensión d, $\boldsymbol{pbest}_{id}$ es el valor en la dimensión d de la partícula con el mejor valor objetivo encontrado por la partícula i, y $\boldsymbol{gbest}_d$ es el valor en la dimensión d del individuo del cúmulo (*swarm*) que encontró el mejor valor objetivo. Como puede observarse en la fórmula, un incremento de ρ_2 sobre ρ_1 aumenta la influencia del valor *gbest*, lo cual resulta en una mayor *exploración* del espacio de soluciones; decrementando el valor de ρ_2 sobre ρ_1 causa que la partícula se mueva en dirección más cercana a *pbest*, lo cual resulta en una mayor *explotación* del espacio. Cuando se habla de *exploración* se considera la habilidad del algoritmo para explorar las diferentes regiones del espacio de búsqueda a fin de encontrar buenas soluciones. Mientras que la *explotación* es la habilidad que el algoritmo posee de concentrarse en una porción del espacio que sea promisoria con el fin de refinar soluciones buenas, y encontrar posiblemente el óptimo.
Además de estas fórmulas, un algoritmo PSO cuenta con los siguientes parámetros:

- $\boldsymbol{N}$: número de partículas involucradas en la resolución del problema.

- ***D***: número de variables del problema, equivalente al número de dimensiones de una partícula.
- ***Xmax*** y ***Xmin***: parámetros que limitan el área de búsqueda.
- ***Vmin*** y ***Vmax***: parámetros opcionales que limitan la velocidad de las partículas. Son opcionales porque si bien es necesaria una restricción de los valores que la velocidad puede alcanzar, existen otros métodos como el factor de constricción que cumplen con la finalidad de limitar dichos valores.
- ***Condición de parada del algoritmo***: puede usarse un *THRESHOLD* que establece un nivel de error aceptable entre el ideal y el óptimo obtenido, una cantidad máxima de iteraciones (*ciclos de vuelo*) o alguna otra que se prefiera.

El pseudo código de *Particle Swarm Optimization* se muestra en el Algoritmo 3.1.

Algoritmo 3.1 PSO básico

Fase de Inicialización del cúmulo:
Para cada partícula i, $i \in [1, N]$ **hacer**
  Para cada dimensión d, $d \in [1, D]$ **hacer**
    Aplicar a x_{id} un valor aleatorio en el rango $[Xmin, Xmax]$
    Aplicar a $pbest_{id}$ el valor de x_{id}
    Aplicar a v_{id} un valor aleatorio en el rango $[Vmin, Vmax]$
  Fin Para
  Calcular el $fitness\_x_i$
  Aplicar a $valor\_fitness\_pbest_i$ el valor $fitness\_x_i$
  Aplicar a $gbest$ el valor de x_i si el $fitness\_x_i$ es mejor que $valor\_fitness\_gbest$
Fin Para
Fase de Búsqueda:
Mientras no se alcance la condición de parada **hacer**
  Para cada partícula i, $i \in [1, N]$ **hacer**
    Para cada dimensión d, $d \in [1, D]$ **hacer**
      Calcular v_{id}
      Calcular x_{id}
    Fin Para
    Calcular $fitness\_x_i$
  Fin Para
  Actualizar $gbest$ con x_i si valor $fitness\_x_i$ es mejor que $valor\_fitness\_gbest$
  Actualizar $pbest\_i$ y $valor\_fitness\_pbest_i$ si $fitness\_x_i$ es mejor que $pbest\_i$
Fin Mientras
Retornar Resultados

Observando el Algoritmo 3.1, la fase de *Inicialización* del cúmulo asigna a cada dimensión de cada partícula de la población un valor aleatorio en el rango establecido por $[Xmin, Xmax]$. Este rango depende exclusivamente del problema que se pretende resolver con la heurística. Lo mismo sucede con los vectores de velocidad (uno por cada partícula),

los que son inicializados en un rango [$Vmin$, $Vmax$] (líneas 1 a 7). Los valores objetivo son calculados y almacenados (línea 8). A cada partícula *pbest* se le copia el valor de cada dimensión de su individuo correspondiente, al igual que la partícula *gbest* a la que se le asigna el individuo con mejor valor objetivo (líneas 9 a 11).
La segunda y más importante es la fase de *búsqueda*, en la cual el cúmulo (*swarm*) recorre el espacio para hallar la mejor solución posible. En un proceso repetitivo (y hasta que se cumpla una condición de parada), se actualizan las fórmulas de aprendizaje (velocidad y posición) con base en los valores actuales de *pbest* y *gbest* (líneas 12 a 18). Los valores de *fitness* son recalculados y utilizados para reemplazar las partículas *pbest* y *gbest*, si es que en la presente iteración se obtuvieron mejores valores objetivo (líneas 19 a 23). La última tarea consta del reporte de los resultados obtenidos (línea 24).

3.2.4. Vecindarios en PSO

Una de las principales características que posee PSO es la interacción social que se observa entre los individuos. Por eso es importante el estudio de las posibles estructuras sociales que pueden incluirse en PSO. Todos los individuos pertenecientes a la misma estructura social comparten información, aprenden y siguen a los mejores dentro de su propia estructura.
Investigaciones que datan de 1940 han mostrado que la comunicación entre los individuos de un grupo o población así como su desempeño (*performance*), están íntimamente ligados a la estructura social que posee el grupo. Así también, en *Particle Swarm Optimization*, dichas interacciones se observan particularmente entre los vecinos inmediatos adyacentes, y son estas interacciones las que provocan que el algoritmo "funcione", ya que cada partícula por sí misma no podría evolucionar demasiado. Una pregunta obvia sería: ¿cuál es la mejor estructura social que permite la interacción óptima entre partículas? Una analogía entre la población de partículas y una población de humanos podría ser la respuesta [84].
La estructura más simple que se puede plantear es aquélla donde todos los individuos son influenciados exactamente de la misma forma, por la mejor solución encontrada por algún miembro de la población. Este tipo de modelo recibe el nombre de *gbest* (mejor global), y es equivalente a una estructura donde todos los individuos están conectados entre sí.
Otra estructura que surge naturalmente recibe el nombre de *lbest* (mejor local). En ésta cada individuo es afectado por el mejor desempeño de sus k vecinos inmediatos en la "población topológica" (comúnmente llamada *vecindario*).
La elección de la estructura social es uno de los principales inconvenientes que presenta la implementación de PSO, debido a la importancia y fuerte influencia que ésta ejerce en el desempeño del algoritmo.
La primera decisión involucra la elección del modelo: ¿*gbest* o *lbest*? El modelo *gbest* tiende a converger prematuramente, debido a la pérdida de la diversidad. El modelo *lbest* es más lento de converger, pero conserva la diversidad [84]. Los resultados obtenidos con algoritmos que incluyen vecindarios suelen ser, generalmente, mejores cuando son comparados con los obtenidos con modelos *gbest*.
Varias estructuras *lbest* pueden ser utilizadas. James Kennedy en 1999 [83] presentó al-

gunas estructuras simples:

- **Circular o Anular**: cada individuo está conectado con sus k vecinos inmediatos, intentando imitar al mejor de ellos. En la figura 3.2 a) puede observarse un ejemplo con $k = 2$. En este tipo de estructura, cada vecindario se encuentra solapado para facilitar el intercambio de información y, de esta manera, converger a una única solución al final del proceso de búsqueda. La información entre vecindarios fluye lentamente, lo cual provoca que la convergencia sea más lenta pero al mismo tiempo más segura. En funciones multimodales (aquéllas que poseen varios óptimos) este tipo de estructura suele obtener un buen desempeño con respecto a otras estructuras. Muchas variantes de este tipo de estructura pueden ser creadas, en donde la conexión de las partículas puede ser elegida aleatoriamente o siguiendo algún criterio, dependiendo del problema. La figura 3.2 b) muestra un ejemplo.

- **Rueda**: un único individuo (el central) está conectado a todos los demás, y todos éstos están conectados solamente al individuo central. La figura 3.2 c) ilustra el concepto. En esta estructura una única partícula sirve de punto focal, y toda la información entre los individuos es canalizada por ese punto. La partícula focal compara el desempeño de todas las partículas en el vecindario para ajustar las posiciones a la mejor. Tan pronto como la mejor posición sea detectada, la información es enviada a todas las partículas para que efectúen el cambio. La estructura de rueda propaga las buenas soluciones en forma demasiado lenta lo cual puede no ser del todo bueno, porque el proceso de búsqueda se tornaría demasiado lento. Algunas variantes pueden ser introducidas, como desconectar algunas partículas de la partícula focal y conectarlas a otras partículas en el vecindario (figura 3.2 d).

- **Arcos Aleatorios**: para N partículas, N conexiones simétricas son asignadas entre pares de individuos. Como su nombre lo indica, las conexiones son asignadas de forma aleatoria entre las N partículas. La figura 3.2 e) muestra esta estructura para un tamaño de población de 12 individuos.

- **Pirámide**: la idea de esta topología es la creación de una estructura tridimensional que comunique a las partículas. Se asemeja a un modelo de alambre (*wire-frame*) tridimensional, como se observa en la figura 3.2 f).

- **Toroidal**: introducida por [55] esta topología puede utilizarse como estructura de vecindario. Es una superficie cerrada con base en la topología de malla (figura 3.2 g)) pero a diferencia de esta última donde cada partícula posee dos vecinos, en la toroidal posee 4 vecinos directos (figura 3.2 h)).

Cada una de estas topologías, a su vez, puede variar algunas de sus conexiones, para constituir subvecindarios. Lo importante es lograr una topología que mejore el desempeño del algoritmo. Actualmente, no existe una topología que sea buena para todo tipo de problema. Generalmente, las estructuras completamente conectadas son mejores para problemas unimodales (un único óptimo), mientras que las menos conectadas funcionan mejor para problemas multimodales, dependiendo del grado de interconexión entre las

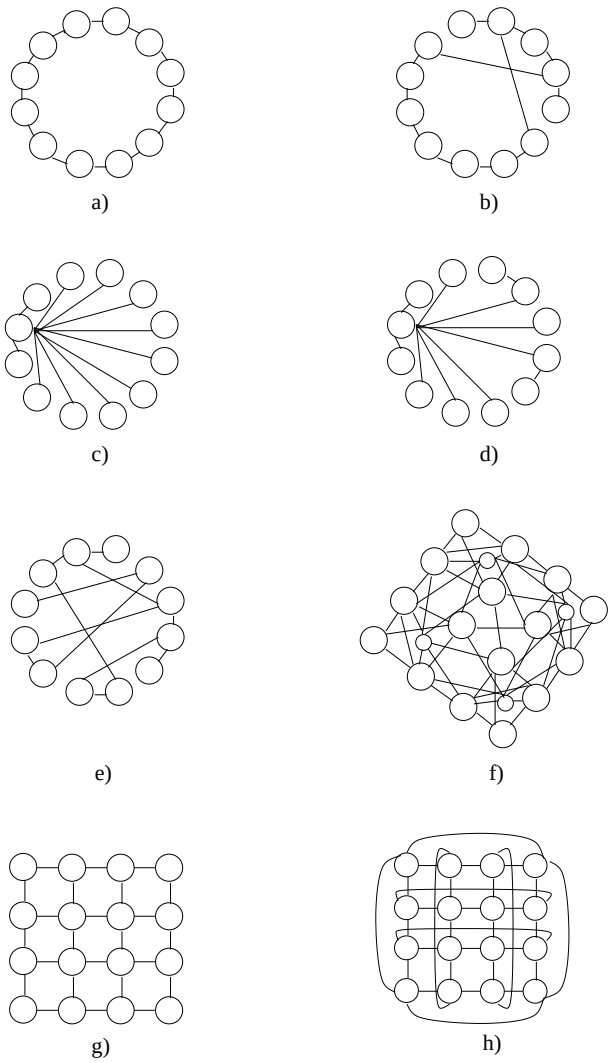


Figura 3.2: (a) Topología circular con $k = 2$. (b) Topología circular con conexiones variantes y 2 subvecindarios. (c) Topología de rueda. (d) Topología de rueda con conexiones variantes. (e) Topología de arcos aleatorios. (f) Topología piramidal. (g) Topología de malla. (h) Topología toroidal.

partículas [83] [142].
Normalmente, los vecinos de las partículas son determinados con base en los índices de las mismas (o posición) dentro de la estructura de datos en la que se encuentran almacenadas. Por ejemplo, para un vecindario cualquiera de *lbest*, con número de vecinos $k = 2$, la partícula i tendrá como vecinos a las partículas $i - 1$ y a la $i + 1$, es decir, a las dos que se encuentran a su lado en la estructura de datos. Algunas variantes se basan en la distancia euclidiana (entre posiciones de partículas) para el cálculo de los vecinos.

El objetivo de utilizar estructuras de vecindarios es evitar la convergencia prematura del algoritmo hacia óptimos locales. Esto es posible porque en una topología de vecindario, cada individuo es influenciado por el mejor valor encontrado por grupos más pequeños de

partículas, y no por el mejor valor hallado por algún individuo de la población entera.
Con una estructura de vecindario, el algoritmo tiende a explorar mejor el espacio de búsqueda y, de esta manera, efectúa una convergencia más lenta posibilitando aumentar la calidad de las soluciones encontradas.

3.3. Diferentes versiones de PSO

Los resultados empíricos presentados en la literatura muestran la habilidad de PSO para resolver una gran cantidad de problemas de optimización [145], aunque otros ilustran que la heurística presenta problemas en la convergencia a buenas soluciones en algunos problemas [110] [14]. Por tal motivo, se han propuesto varias modificaciones al PSO básico, de manera que se mejore la velocidad de convergencia y la calidad de las soluciones encontradas. A continuación se describen brevemente algunas variantes del algoritmo original.

3.3.1. Evolución de PSO desde espacios continuos a espacios discretos

El algoritmo PSO fue creado originalmente para tratar problemas de optimización en espacios de búsqueda continuos y los resultados que se obtuvieron entonces, indicaron que el algoritmo era efectivo y eficiente. El PSO original busca soluciones en espacios "infinitos" simbolizados como $\mathbb{R}^n$, es decir, en el espacio $n-$dimensional de números reales.
En la práctica existen muchos problemas de optimización que involucran variables discretas o binarias (espacios discretos) tales como los problemas de planificación (*scheduling*) y de ruteo (*routing*).
Realizar el cambio de espacio de trabajo de un algoritmo, muchas veces puede resultar fácil de solucionar en algunos aspectos, pero difícil en otros. Muchos problemas surgen en la adaptación de las fórmulas (3.1) y (3.2) (cambio de trayectoria y actualización de la partícula), creadas para espacios continuos. En muchos casos cambia también el significado de lo que las fórmulas representan. Otros problemas surgen también en la codificación de las partículas (posibles soluciones).

En el año 1997, Kennedy y Eberhart [40] definieron la primera versión discreta binaria de PSO. En ésta, las partículas se codifican como cadenas binarias. La fórmula de actualización de velocidades se describe de forma similar a la fórmula (3.1), pero limitando $(\rho_1 * r_1) + (\rho_2 * r_2)$ a un valor que sea 4 o menor. La fórmula (3.2) de actualización de partículas cambia y se reemplaza por una donde las dimensiones de cada partícula resultan en ceros o unos. Para tal fin se utiliza la función *sigmoidal s* (figura 3.3), la cual es interpretada como "cambios de probabilidad" [84]. Esta probabilidad es la de encontrar una mejor región, más que un cambio de dirección o trayectoria. Luego, la actualización de una partícula binaria se realiza utilizando la función s que se describe en la fórmula (3.3):

$$s(v_{id}) = \frac{1}{1 + exp(-v_{id})} \tag{3.3}$$

la cual toma como parámetro la velocidad de la $i-$ésima partícula v_{id} para actualizar la partícula i. Con valores de v_{id} altos, la función tiende a seleccionar el valor 1, mientras que si los valores son bajos la función tiende a favorecer la elección de 0. Una buena práctica es limitar la velocidad al rango [-4, 4] de forma tal que la función sigmoidal no se aproxime demasiado a los valores extremos de cada dimensión (0 y 1), y de esta forma favorecer a que se produzca un cambio en la dimensión.

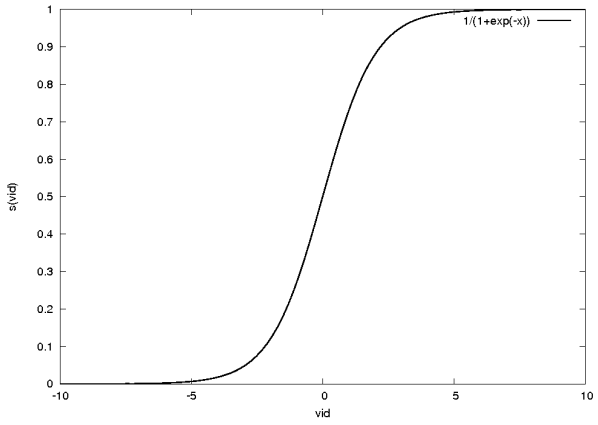


Figura 3.3: Función Sigmoidal

Utilizando esta función, la fórmula (3.2) es reemplazada por la fórmula (3.4):

$$si \;\; \rho_{id} < s(v_{id}) \;\; entonces \;\; x_{id} = 1; \;\; en \;\; otro \;\; caso \;\; x_{id} = 0 \tag{3.4}$$

siendo ρ_{id} la dimensión d de un vector binario con números aleatoriamente generados.

Posteriormente, Mohan y Al-kazemi [125] propusieron varias aproximaciones binarias de PSO: directa, por quantum, de regularización, mezclada, etc., aunque no muchas conclusiones lograron deducirse de estos estudios.

Hu, Eberhart y Shi introdujeron en el año 2003 un PSO para tratar problemas con permutaciones [74], específicamente trataron el problema de las *n-reinas*. En su trabajo, los autores codificaron las partículas como permutaciones de un grupo de valores únicos (sin repetición). Las velocidades fueron redefinidas (vector V) basadas en la similitud entre dos partículas. La fórmula de actualización se transformó en un procedimiento de intercambios aleatorios dependiendo de las velocidades asociadas a cada partícula. Gráficamente puede observarse un ejemplo en la figura 3.4. La idea es seleccionar la dimensión del vector de posiciones (X) que se debe modificar. Para ello se considera el vector de velocidades normalizado $|V|$, así cada dimensión del mismo es un valor entre 0 y 1. Se toma un valor aleatorio en el mismo rango para decidir cuál será la posición dentro del vector

de velocidades. Todas aquellas dimensiones de $|V|$ que superen ese valor aleatorio deberán ser intercambiadas en X. En el ejemplo se tomó el número aleatorio 0.7 entonces la cuarta posición de X es una de las posiciones que será modificada, ya que la cuarta posición de $|V|$ es superior a 0.7. Pero, ¿con qué otro valor se intercambia la cuarta dimensión de X? Se toma el valor de la cuarta dimensión del vector *pbest* y se la busca en X. Es seguro que ese valor se encontrará en X ya que se trabaja con permutaciones. En *pbest* el 5 es el que se encuentra en la cuarta posición así es que se busca ese valor en el vector X. El valor 5 está en la primera posición, entonces se intercambian los valores de las posiciones 4 y el de la posición 1, obteniendo así el nuevo vector de posiciones X, es decir, $X + V$ (los autores utilizan esa notación para nombrar al nuevo vector de posiciones). Los autores también introdujeron un factor de mutación para prevenir los estancamientos de los vectores *pbest* en óptimos locales. Estudios preliminares de esta versión aplicada al famoso problema de las *n-reinas*, mostraron que es promisoria en la resolución de problemas discretos.

En [15] se presentó una versión discreta de PSO (denominada CLUDIPSO) aplicada a la resolución de *clustering* de documentos cortos. Cada partícula del algoritmo representa un agrupamiento válido (una dimensión por documento a clasificar) y está definida como un vector de números enteros en el rango $[1, n]$ siendo n el número de grupos. La fórmula de actualización de velocidades es similar a la (3.1) (modelo global) mientras que la ecuación (3.2) de actualización de partícula se modificó asignándole a cada individuo su *pbest* correspondiente. La actualización de la partícula no se efectúa en todas las dimensiones, en todas las iteraciones del algoritmo sino que está supeditada al valor de su vector de velocidad correspondiente. El procedimiento de decisión es el siguiente: 1) todas las dimensiones del vector de velocidades se normalizan en el rango [0, 1]; 2) se calcula un número aleatorio r entre 0 y 1; 3) todas las dimensiones que en el vector de velocidades sean mayores a ese r se seleccionan en el vector de posiciones y se reemplazan por su valor *pbest* correspondiente. CLUDIPSO cuenta con un operador de mutación dinámica para evitar el problema de convergencia prematura. Dicho operador se actualiza en cada iteración del algoritmo.

3.3.2. PSO con Control de Velocidad

Uno de los objetivos fundamentales de toda heurística es lograr un balance entre la explotación y exploración del espacio de búsqueda. En PSO esta tarea recae en la velocidad de las partículas. Si no se controla este elemento, podría suceder que las partículas escapen del espacio de búsqueda del problema, resultando esto en la divergencia de los individuos. Por este motivo es común limitar el crecimiento de la velocidad [42] utilizando algún método. El más simple es emplear un parámetro de velocidad máxima (elegido dependiendo el problema), de manera que la actualización se efectúe siguiendo la fórmula (3.5) en vez de la (3.1):

$$v_{id} = \begin{cases} v'_{id} & si\ v'_{id} < Vmax \\ Vmax & si\ v'_{id} \geq Vmax \end{cases} \tag{3.5}$$

donde v_{id} será la nueva velocidad, definida como: v'_{id} (velocidad calculada con la ecuación (3.1)) o $Vmax$, dependiendo de la condición. El valor elegido para $Vmax$ es muy impor-

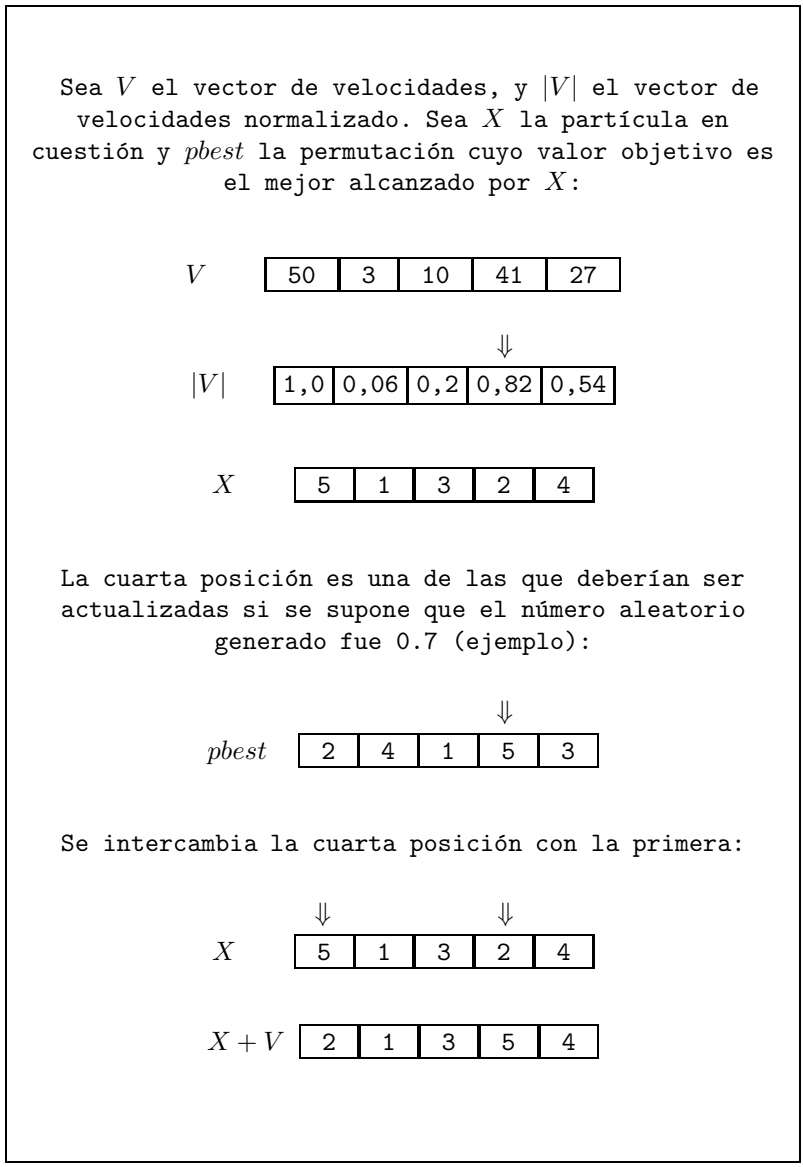


Figura 3.4: Actualización de una partícula en PSO para permutaciones

tante, ya que controla la granularidad de la búsqueda escalando las velocidades. Valores grandes facilitan la exploración del espacio, aunque si los valores son muy grandes se podría perder alguna región (exploración excesiva). Los valores más chicos aseguran la explotación, aunque si son demasiado pequeños podría suceder que el algoritmo no explorara bien, dejando sin visitar mejores regiones (encuentra óptimos locales). Además, con valores reducidos de velocidad se necesitarían más ciclos para que el algoritmo alcanzara buenos valores, ya que la búsqueda es más específica.
Por ello es importante una buena selección de *Vmax*. Lo ideal sería un valor que asegure el balance de ambos objetivos: exploración y explotación. Una opción es fijar el valor máximo de la velocidad de cada dimensión en una fracción del dominio de cada dimensión

del espacio de búsqueda. Esto es:

$$Vmax_j = \delta(Xmax_j - Xmin_j)$$

donde $Vmax_j$ es la máxima velocidad permitida para la $j-$ésima dimensión de cada partícula, y $[Xmax_j, Xmin_j]$ el rango de valores válidos para la $j-$ésima dimensión. El valor δ es un entero en el rango $(0, 1]$ (aunque para cada problema podría variar [171]), el cual será asignado a $Vmax_j$.
Una situación común de esta variante es que en determinado momento de la búsqueda, la mayoría o todas las velocidades pueden contener el valor máximo. Si esto no es tratado, todas las partículas se encontrarán en los límites del hipercubo $[x_i - Vmax, x_i + Vmax]$, provocando así que no se exploren áreas donde se encuentra el óptimo. Para solucionar esto, una posibilidad es la incorporación de un factor de inercia como se describirá en la próxima subsección. Otra forma es decrementar el valor de $Vmax$ a medida que transcurren los ciclos de vuelo. La idea es que al comienzo del proceso de búsqueda el algoritmo explore grandes áreas del espacio, pero a medida que las iteraciones transcurren, la velocidad se decrementará para favorecer la explotación de esas zonas.

3.3.3. PSO con Factor de Inercia

En 1998, Shi y Eberhart [169] [170] introdujeron el factor de inercia (w) en la fórmula de actualización de la velocidad de las partículas. Inicialmente surgió como mecanismo de control de las habilidades de exploración y explotación del cúmulo. También para que la velocidad no exceda los límites establecidos. Numerosos estudios concluyeron que como mecanismo de control de exploración/explotación funciona correctamente, aunque no logró eliminar completamente la necesidad del chequeo de la velocidad. El factor multiplica a la velocidad previa en la fórmula básica de velocidad y, puede permanecer fijo o ser linealmente decrementado en cada ciclo de vuelo. Luego, la nueva velocidad es actualizada de la misma manera que en las otras variantes de PSO. Un factor de inercia con valor de 1 introduce una preferencia para que las partículas continúen moviéndose en la misma dirección. De otro modo, se cambia la dirección. La ecuación (3.1) de velocidad se reemplaza por la (3.6), mientras que la de actualización de partículas (fórmula (3.2)) permanece sin cambios en esta variante.

$$v_{id} = w * v_{id} + \rho_1 * r_1 * (\boldsymbol{pbest}_{id} - x_{id}) + \rho_2 * r_2 * (\boldsymbol{gbest}_d - x_{id}) \tag{3.6}$$

El valor de w es importante para la convergencia del algoritmo. Para valores $w \geq 1$ las velocidades se van incrementando en cada ciclo de vuelo acelerando las partículas al máximo valor permitido, lo que produce la divergencia del cúmulo (incremento de la diversidad). Para valores $w < 1$, las partículas se desaceleran hasta velocidad casi 0 (explotación). Generalmente el mejor valor para w es dependiente del problema [171].
Las primeras versiones de PSO con factor de inercia utilizaban un valor constante durante toda la búsqueda. Estudios posteriores demostraron que para algunos problemas, el decremento del factor w en el tiempo permite combinar la exploración de una búsqueda global con la explotación de una búsqueda local (w dinámico). Típicamente para el

caso dinámico, w es reducido linealmente utilizando 2 valores límites: w_{start} y w_{end} que generalmente toman los valores: 0.9 y 0.4. Luego, mediante la fórmula (3.7), el factor de inercia es actualizado en cada paso de tiempo t hasta el final de iteraciones (T_{max}).

$$w(t) = \frac{(T_{max} - t) * (w_{start} - w_{end})}{T_{max}} + w_{end} \tag{3.7}$$

Una variante utiliza la forma contraria de actualización del factor de inercia, es decir, el factor se incrementa en cada unidad de tiempo [206] partiendo del valor máximo hasta llegar al mínimo.
Otra posibilidad es la reinicialización aleatoria de w en cada iteración utilizando, por ejemplo, un número aleatorio generado con una distribución Gaussiana:

$$w \sim N(0.72\ ;\ \sigma)$$

con σ lo suficientemente pequeño como para asegurar que w sea menor que 1.
Los decrementos no lineales de w pueden ser utilizados en espacios de búsqueda suaves, ya que permiten la exploración en un lapso corto de tiempo y dedican más tiempo a la explotación. Una posible ecuación para llevar a cabo esta actualización es [191]:

$$w(t+1) = \alpha w(t')$$

donde comúnmente α =0.975 y t' es el paso de tiempo en el que el factor de inercia cambió por última vez. Los autores modifican el factor cuando no existe una mejora en el *fitness*.

3.3.4. PSO con Factor de Constricción

Esta versión fue presentada por Clerc [24] en el año 2000. Este factor consta de un conjunto de complicadas ecuaciones lineales que, al menos teóricamente, derivan en la convergencia óptima de las partículas. En este modelo, la velocidad está restringida por una constante $\mathcal{X}$, con el objeto de asegurar la convergencia a un punto estable del espacio, sin la necesidad de controlar que la velocidad exceda un valor máximo. La ecuación (3.1) de velocidad se reemplaza por la (3.8).

$$v_{id} = \mathcal{X}[v_{id} + \rho_1 * r_1 * (\boldsymbol{pbest}_{id} - x_{id}) + \rho_2 * r_2 * (\boldsymbol{gbest}_d - x_{id})] \tag{3.8}$$

donde el factor $\mathcal{X}$ se define como lo muestra la ecuación (3.9).

$$\mathcal{X} = \frac{2\mathcal{K}}{|2 - \phi - \sqrt{\phi(\phi - 4)}|} \tag{3.9}$$

siendo:

$$\phi = \phi_1 + \phi_2$$

$$\phi \geq 4$$

$$\phi_1 = \rho_1 * r_1$$

$$\phi_2 = \rho_2 * r_2$$
$$\mathcal{K} \in [0, 1]$$

El parámetro $\mathcal{K}$ en la ecuación (3.9) controla la exploración y explotación del cúmulo. Para valores de $\mathcal{K}$ cercanos a 0, se obtiene una rápida convergencia con explotación local. Para valores de $\mathcal{K}$ cercanos a 1, la convergencia se produce más lentamente y se experimenta un grado mayor de exploración. Aunque el valor de $\mathcal{K}$ es constante, algunas variaciones comienzan con valores altos y, a medida que transcurren los ciclos, se va decrementando a fin de aumentar la explotación.
Aunque este factor es bastante similar al de inercia, el de constricción presenta algunas ventajas:

- No es necesario controlar que la velocidad se mantenga inferior a un valor máximo.
- El modelo de constricción garantiza la convergencia con los parámetros antes indicados.
- Los cambios de dirección son regulados por las constantes ϕ_1 y ϕ_2 del factor de constricción.

3.3.5. PSO Completamente Informado

Esta variante de PSO fue propuesta por Mendes et al. [118] en 2004. Cada individuo recibe información completa de todos sus vecinos topológicos. Los autores se basaron en el modelo que utiliza el factor de constricción, argumentando que el valor de ϕ no debería ser dividido en sólo dos valores, sino que se deberían tomar en cuenta todos los individuos del vecindario y, que el valor ϕ sea proporcional a éstos. Ellos propusieron que para una partícula dada, el ϕ debe ser descompuesto en:

$$\phi_k = \frac{\phi}{|\mathcal{N}|} \quad \forall k \in \mathcal{N}$$

donde $\mathcal{N}$ es el vecindario de la partícula. Luego, la ecuación (3.8) se transforma en la (3.10).

$$v_{id} = \mathcal{X}[v_{id} + \sum_{k \in \mathcal{N}} \phi_k * \mathcal{W}(k) * r_k * (\boldsymbol{pbest}_{kd} - x_{id})] \tag{3.10}$$

siendo $\mathcal{W}$ una función de peso establecida previamente.

3.3.6. PSO Jerárquico

Presentado por Janson et al. [80], consiste de un algoritmo que dinámicamente adapta la topología de la población. Los individuos se encuentran organizados en una estructura de árbol, donde aquellos con mejor valor de *fitness* se encuentran en los nodos superiores. En cada iteración las partículas hijas actualizan las velocidades considerando su *pbest* y el mejor valor obtenido por su padre. Luego, el desempeño de los hijos se compara con el de los padres, y en caso de que el primero sea mejor que el de sus progenitores, se intercambian las posiciones de los individuos en la jerarquía del árbol. El grado de ramificación que se utilice para el árbol influye en el balance de la exploración-explotación del algoritmo.

3.3.7. Modelo Puramente Cognitivo

Este modelo provee una tendencia a que las partículas regresen a las mejores posiciones alcanzadas en su pasado, ya que excluye la componente social de la fórmula (3.1) de actualización de velocidad. En esta versión, la ecuación se transforma en la (3.11).

$$v_{id} = v_{id} + \rho_1 * r_1 * (\boldsymbol{pbest}_{id} - x_{id}) \tag{3.11}$$

El factor de inercia puede ser incluido en la ecuación (3.11) si se considera necesario. En [82] se reportó que este modelo es levemente más vulnerable a las fallas, comparado con el modelo completo (fórmula (3.1)). Tiende a explotar áreas locales donde las partículas fueron inicializadas. El número de iteraciones que requiere es mayor que el modelo completo y, falla cuando los coeficientes de aceleración y velocidad son valores pequeños.

3.3.8. Modelo Puramente Social

En el modelo completamenete social se excluye de (3.1) la componente cognitiva resultando en la fórmula (3.12).

$$v_{id} = w * v_{id} + \rho_2 * r_2 * (\boldsymbol{gbest}_d - x_{id}) \tag{3.12}$$

El valor *gbest* puede ser reemplazado por *lbest* si se utilizan vecindarios. En esta versión las partículas siguen la atracción producida por el mejor del cúmulo o el mejor del vecindario. No considera de manera directa la posibilidad de retornar a un lugar visitado previamente. En [82] Kennedy mostró que este modelo es más rápido y efectivo que el modelo cognitivo y, en algunos casos, que el modelo completo.

3.3.9. Actualización asincrónica

El Algoritmo 3.1 realiza las actualizaciones de manera sincrónica, es decir, la actualización de los mejores valores encontrados (memorias) se realiza en forma separada de la actualización de las posiciones de las partículas. Otra opción es realizar la actualización de la posición de cada individuo, e inmediatamente actualizar el valor de los mejores encontrados (si correspondiera). El Algoritmo 3.2 muestra un pseudo código de una versión de PSO asincrónico, en donde la única diferencia con el Algoritmo 3.1 yace en la ubicación de las actualizaciones de *gbest* y *pbest* (líneas 20 y 21 del Algoritmo 3.2).

El modelo asincrónico mantiene actualizadas instantáneamente a las mejores partículas del cúmulo, mientras que en la versión sincrónica esta actualización se realiza sólo una vez por iteración. En [19] los autores concluyeron que las actualizaciones asincrónicas son más importantes cuando se utiliza el modelo *lbest*, ya que el cambio de información inmediato es beneficioso en los vecindarios. También reportaron que las actualizaciones sincrónicas son mejores cuando se utiliza el modelo *gbest*.

Algoritmo 3.2 PSO asincrónico.

Fase de Inicialización del swarm:
Para cada partícula i, $i \in [1, N]$ **hacer**
  Para cada dimensión d, $d \in [1, D]$ **hacer**
    Asignar a x_{id} un valor aleatorio en el rango $[Xmin, Xmax]$
    Asignar a $pbest_{id}$ el valor de x_{id}
    Asignar a v_{id} un valor aleatorio en el rango $[Vmin, Vmax]$
  Fin Para
  Calcular el $fitness\_x_i$
  Asignar a $valor\_fitness\_pbest_i$ el valor de $fitness\_x_i$
  Asignar a $gbest$ el valor de x_i si el $fitness\_x_i$ es mejor que $valor\_fitness\_gbest$
Fin Para
Fase de Búsqueda:
Mientras no se alcance la condición de parada **hacer**
  Para cada partícula i, $i \in [1, N]$ **hacer**
    Para cada dimensión d, $d \in [1, D]$ **hacer**
      Calcular v_{id}
      Calcular x_{id}
    Fin Para
    Calcular $fitness\_x_i$
    Actualizar $gbest$ con x_i si valor $fitness\_x_i$ es mejor que $valor\_fitness\_gbest$
    Actualizar $pbest\_i$ y $valor\_fitness\_pbest_i$ si $fitness\_x_i$ es mejor que $pbest\_i$
  Fin Para
Fin Mientras
Retornar Resultados

3.4. Breve estado del arte: aplicaciones de PSO

La heurística PSO ha sido utilizada en un sinnúmero de aplicaciones y en todas con muy buen desempeño. Su robustez hace a esta heurística prometedora para resolver problemas de distintas características, con diferente cantidad de variables y en un tiempo bastante aceptable. Año tras año se emplea la heurística para nuevas aplicaciones por lo que en las siguientes subsecciones se realiza una breve descripción de los trabajos más relevantes reportados en la literatura especializada, a la fecha.

3.4.1. Redes Neuronales

Esta es una de las primeras aplicaciones de PSO. Se utilizó la heurística en el entrenamiento de redes neuronales *feed-forward* [39] [82]. Los estudios preliminares demostraron que PSO era eficiente en el entrenamiento de estas redes particulares, y fue así que posteriormente se utilizó para cualquier arquitectura de redes. Los resultados eran rápidos y seguros, y el algoritmo funcionaba bien independientemente si el aprendizaje de las redes fuera supervisado o no. También se empleó PSO en la selección de la arquitectura de redes como se muestra en [202].
Estas redes neuronales fueron utilizadas en problemas reales como el análisis de temblores

en las personas para la detección del Mal de Parkinson [41] o, para estimar el estado de carga de baterías con el fin de evitar la sobrecarga de las mismas [143]. El modelo PSO con *lbest* fue utilizado para el entrenamiento de redes neuronales empleadas en la estabilización de controladores en sistemas de potencia [69] (electricidad, por ejemplo). Otras aplicaciones incluyen el entrenamiento de redes para predecir niveles de polución [113], problemas de tráfico [175], y juegos de dos personas [119].

3.4.2. Aprendizaje en Juegos

PSO ha sido utilizado para coevolucionar agentes en juegos de probabilidades, de suma cero (por ejemplo el *tic-tac-toe* [119]), de suma no-cero (el Dilema del Prisionero [52]) y basados en tablero (como ajedrez o ubicación de $n-$reinas [74] en un tablero de $n \times n$ posiciones).

3.4.3. Aplicaciones en *Clustering*

La tarea de *Clustering* o agrupamiento tiene como objetivo la formación de grupos de elementos de datos que poseen alguna similitud. Esa similitud generalmente se mide con base en la distancia euclidiana que existe entre los elementos. Luego, el problema de *clustering* se reduce a uno de optimización. Entre las aplicaciones de PSO para agrupamiento se puede encontrar la clasificación de imágenes [133], y de datos en general [184]. Tillett [179] aplicó PSO para el agrupamiento de nodos en redes *ad hoc*. En [77] se utilizó PSO para realizar el agrupamiento de textos cortos con muy buenos resultados comparados a los de otras técnicas del estado del arte de *Clustering*.

3.4.4. Aplicaciones en Diseño

PSO es empleado en un gran número de aplicaciones relacionadas al diseño. Por ejemplo, el diseño de: alas de aviones [190], antenas [57], circuitos [28], estabilizadores de sistemas de potencia [1], sistemas concurrentes óptimos [204], piezas mecánicas [16] y, armaduras [166] (diseño estructural).

3.4.5. Programación de Horarios y Planeación de Tareas

La programación de horarios y planeación de actividades son tareas cotidianas que están presentes en un sinnúmero de aplicaciones. Algunas de ellas incluyen: planeamiento de la ruta que debe seguir un avión [168], despacho en sistemas de potencia [44] (electricidad, por ejemplo), asignación de tareas [161] (actividades industriales u otras áreas), problema del viajante [135] y, control de válvulas en la combustión interna de motores [152].

3.4.6. Nuevas aplicaciones

Un compendio bastante completo de aplicaciones modernas de PSO puede ser encontrado en [145]. En él se incluyen campos como la biología, medicina y farmacéutica:

optimización de movimientos biomecánicos en humanos, clasificación de tipos de cáncer, predicción de supervivencia, selección de biomarcadores, predicción de estructuras de proteínas y análisis de electroencefalogramas. En control: flujo de tránsito, de motores ultrasónicos, en sistemas de potencia (eléctricos), de combustión, aterrizaje automático aéreo. En ingeniería automotriz: control de torque, control de vehículos eléctricos e híbridos, control de velocidad, optimización de la combustión interna de motores y optimización de sistemas de propulsión eléctrico-nuclear. También existen aplicaciones en el área de detección y recuperación frente a fallas de sistemas, en el campo financiero, computación gráfica, visualización, imagen, video, metalurgia y robótica. En seguridad militar se ha utilizado PSO para detección de intrusos, criptografía y criptoanálisis, asignación de armamento y optimización de la efectividad de misiles.

3.5. Optimización Evolutiva versus PSO

Muchos investigadores consideran que la heurística PSO está incluida en el paradigma de Computación Evolutiva (CE). Lo cierto es que PSO toma ideas de varias disciplinas: Inteligencia Artificial, Computación Evolutiva y Teoría Colectiva, con lo cual cualquier encasillamiento específico en alguna de éstas, sería inadecuado. En [46] se mantiene la opinión de que entre PSO y los Algoritmos Evolutivos existen algunas similitudes, pero también diferencias que concluyen en que la heurística pertenece a un paradigma separado.
En [4] el autor realiza una distinción filosófica entre ambos métodos de optimización, la cual se resume a continuación.

- Los algoritmos de CE utilizan 3 funciones en el proceso de búsqueda: selección, cruzamiento y mutación. PSO emplea sólo dos de ellas, excluyendo la selección explícita. El objetivo de esta función en CE es el de reubicar los individuos cuyo desempeño en el proceso de búsqueda es pobre, intercambiándolos por una combinación de individuos buenos (los *padres*). En PSO el rol del mecanismo de selección es cubierto por la memoria que cada individuo conserva del mejor valor alcanzado (*pbest*). Ese valor actúa como los *padres* usados en CE con la diferencia de que en PSO no se crean nuevos individuos, sino que se los manipula y adapta. Esto implica que en PSO la información relevante de los *padres* se encuentra contenida directamente en la partícula (por utilizar el factor *pbest*), mientras que en CE esta información no está presente en cada individuo y debe ser compartida explícitamente a través de la función de cruzamiento.

- En PSO la función de mutación (si la hubiera) es una operación altamente direccional (involucra varias direcciones al mismo tiempo). Esta función modificará el vector de velocidades para que la partícula cambie su dirección a una que sea combinación de 2 valores: *pbest* y *gbest* (o *lbest* si se utiliza el modelo local). En cambio, en Optimización Evolutiva, la mutación es un operador unidireccional direccionando al individuo en un único sentido.

- En Optimización Evolutiva existe un control explícito del proceso de mutación variando la severidad en el proceso de búsqueda, ya que los valores de mutación sufren un proceso de optimización similar al de la función de *fitness*. Esto provoca que cuando el óptimo sea detectado en las proximidades de una región, los valores de mutación se ajusten a una búsqueda de granularidad fina. Contrariamente, PSO no posee esta capacidad de detección de óptimo cercano, con lo cual no existe un mecanismo para ajustar el paso de velocidad en determinada región. Como la velocidad no se puede decrementar (a menos que se extralimite de los valores máximos), al final del proceso de búsqueda cuando el algoritmo debería explotar la región, PSO no es capaz de mantener adecuadamente velocidades pequeñas para lograr este efecto.

Capítulo 4

Solución de Problemas Mono-objetivo sin restricciones

En este capítulo se presenta un detallado estudio de una versión de la heurística Particle Swarm Optimization para resolver problemas de optimización mono-objetivo sin restricciones en espacios continuos. Los resultados obtenidos son comparados con los de dos algoritmos representativos del estado del arte. El estudio concluye con un análisis estadístico del desempeño de ambos algoritmos.

Contenido del Capítulo

4.1. Organización del capítulo

La siguiente sección comienza con el estudio de funciones sin restricciones, las cuales fueron resueltas con una versión de Particle Swarm Optimization denominada **Bi-PSO**. Esta propuesta se describe en detalle resaltando las diferencias con el algoritmo básico PSO comentado en el capítulo anterior, junto con la justificación de las mejoras incorporadas. A continuación se muestran los resultados obtenidos con Bi-PSO y se compara el desempeño del algoritmo con uno basado en Evolución Diferencial y otro basado en PSO. El capítulo finaliza con el estudio estadístico de los resultados de los algoritmos evaluados.

4.2. Optimización sin restricciones

El tipo de funciones que se tratan en este apartado tienen la característica de poseer una única limitación: el rango de las variables, es decir, los posibles valores que pueden tomar. Fuera de esto no existe restricción alguna sobre los valores que la función puede tomar. Por ende, la zona factible es el espacio completo de búsqueda.
Si bien parecería que el estudio del comportamiento de algoritmos con funciones sin restricciones es un ejercicio de poca valía, debido a su aparente simplicidad (comparadas con las funciones con restricciones), tal presunción es errónea. Contrario a lo que pudiera pensarse, las funciones de este tipo cuentan con características que resultan de suma utilidad para evaluar el desempeño de los algoritmos de optimización. Entre dichas características se encuentran la alta dimensionalidad (un gran número de variables), y la multimodalidad (la presencia de muchos óptimos locales). Estas características dificultan considerablemente la búsqueda, y vuelven inoperables o muy ineficientes a varias técnicas de programación matemática.

4.3. El algoritmo propuesto: Bi-PSO

Esta versión de PSO extiende al PSO original, presentado en el Algoritmo 3.1. Considerando que investigaciones previas [14] mostraron la dificultad del modelo *gbest* para converger a óptimos globales en problemas complejos, se propone una nueva definición de la fórmula que calcula la velocidad de las partículas. El enfoque propuesto busca explotar las características de los modelos *lbest* y *gbest* de PSO combinándolos en la nueva ecuación (4.1).

$$v_{id} = v_{id} + \rho_1 * r_1 * (\boldsymbol{pbest}_{id} - x_{id}) + \rho_2 * r_2 * (\boldsymbol{gbest}_d - x_{id}) + \rho_3 * r_3 * (\boldsymbol{lbest}_{id} - x_{id}) \quad (4.1)$$

donde v_{id} es el valor de la velocidad de la partícula i en la dimensión d, ρ_1 es la velocidad de aprendizaje cognitivo, ρ_2 es la velocidad de aprendizaje social poblacional, ρ_3 es la velocidad de aprendizaje social en vecindario, r_1, r_2 y r_3 son valores aleatorios uniformemente distribuidos en el rango [0, 1], x_{id} es la posición actual de la partícula i en la dimensión d, $\boldsymbol{pbest}_{id}$ es el valor en la dimensión d de la partícula con el mejor valor objetivo encontrado por la partícula i, $\boldsymbol{gbest}_d$ es el valor en la dimensión d del individuo

del cúmulo (*swarm*) que encontró el mejor valor objetivo y $\boldsymbol{lbest}_{id}$ es el valor en la dimensión d del individuo del vecindario de la partícula i que encontró el mejor valor objetivo. Para asegurar la convergencia sin la necesidad de controlar que la velocidad exceda un valor máximo, se incorporó a la ecuación (4.1) un factor de constricción $\mathcal{X}$, resultando en la ecuación (4.2).

$$v_{id} = \mathcal{X}(v_{id} + \rho_1 * r_1 * (\boldsymbol{pbest}_{id} - x_{id}) + \rho_2 * r_2 * (\boldsymbol{gbest}_d - x_{id}) + \rho_3 * r_3 * (\boldsymbol{lbest}_{id} - x_{id})) \quad (4.2)$$

El factor $\mathcal{X}$ es un valor continuo pequeño cuyo objetivo es decrementar la cantidad de velocidad obtenida con la fórmula (4.1). Aunque en [84] se propone un compleja ecuación para obtener $\mathcal{X}$, aquí se propone utilizar un valor del rango [0.1, 0.99] es decir, un valor similar al del factor de inercia presentado anteriormente. $\mathcal{X}$ es un parámetro del algoritmo que se fija con un valor que permanece inalterable en el transcurso del proceso evolutivo. El detalle de la selección del mismo puede ser consultado en el Capítulo 7.

El algoritmo básico de PSO utiliza una fórmula sencilla para la actualización de las partículas, la cual se describe en la ecuación (3.2). En Bi-PSO se emplea esa fórmula pero con una probabilidad p mientras que con probabilidad $1 - p$ se utiliza una ecuación propuesta por Kennedy [85]. Luego, la actualización de cada dimensión de la partícula i se rige por la fórmula (4.3).

$$\begin{cases} \textbf{Si } (aleatorio < p) \textbf{ entonces} \\ \quad x_{id} = x_{id} + v_{id} \\ \textbf{sino} \\ \quad x_{id} = N\left(\frac{pbest_{id} + lbest_{id}}{2}, |pbest_{id} - lbest_{id}|\right) \\ \textbf{Finsi} \end{cases} \quad (4.3)$$

siendo *aleatorio* un número generado con distribución uniforme en el rango [0, 1], x_{id} la dimensión d de la i-ésima partícula, v_{id} la dimensión d de la partícula i, N el generador de números aleatorios con distribución Gaussiana, y $lbest_i$ la partícula con mejor valor objetivo en el vecindario de la partícula i. La probabilidad p es un parámetro estático del algoritmo y su selección se describe en el Capítulo 7.
El modelo de vecindario implementado se basa en una topología circular en la cual cada individuo está conectado con k vecinos directos. La figura 4.1 ilustra esta topología con un vencindario de tamaño 4. Obsérvese que la mejor partícula del vecindario influencia a los demás vecinos replicando, a través de la superposición de partículas en los vecindarios (ver figura 4.2), la información de los mejores a los demás individuos.

Para determinar el orden de las partículas, y así elegir a los $k/2$ vecinos adyacentes inmediatos que se encuentran a cada lado de cada individuo, se utilizó el ordenamiento que poseen las partículas dentro de la estructura de almacenamiento. Cualquier política podría emplearse para determinar la adyacencia de dos o más individuos, y la utilizada en esta versión es una de las más sencillas. Por ejemplo, si se contara con una población de

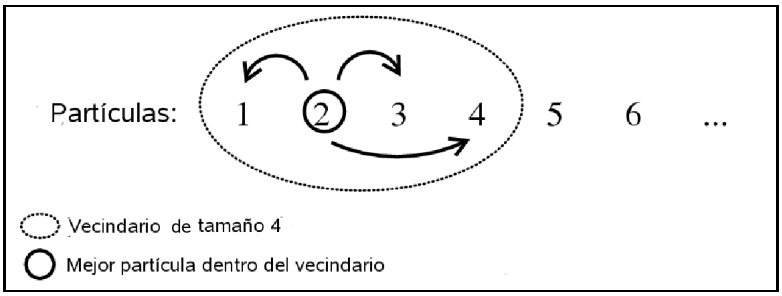


Figura 4.1: Topología circular.

6 partículas y se considerara un vecindario de tamaño 3, los vecindarios posibles serían: (1,2,3), (2,3,4), (3,4,5), (4,5,6), (5,6,1), (6,1,2). La figura 4.2 ilustra esta situación.

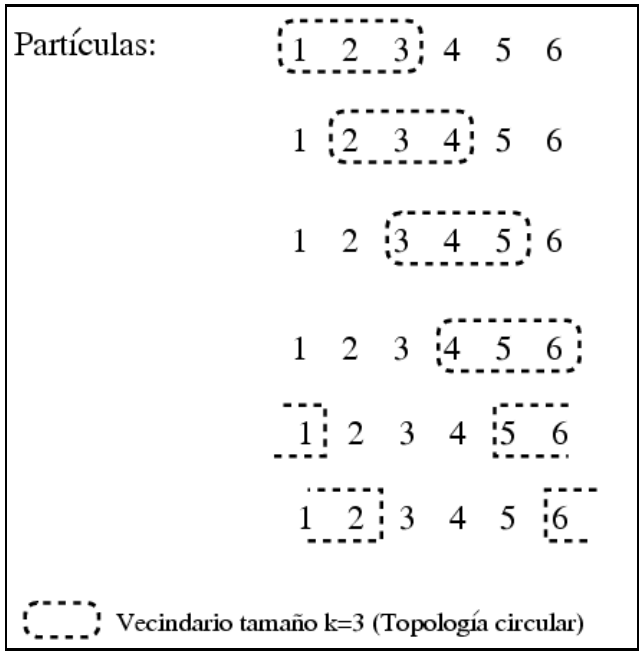


Figura 4.2: Vecindarios para una población de 6 individuos.

Además de las nuevas fórmulas de actualización de velocidades y partículas, un operador de mutación dinámica [14] fue empleado para evitar posibles puntos de convergencia prematura. Este operador se aplica a cada individuo de la población con una probabilidad pm, valor que es calculado considerando la cantidad máxima de ciclos del algoritmo (max_cycle) y el ciclo actual en el momento en el que se aplica ($current\_cycle$). El valor que pm puede tomar se encuentra limitado al rango [min_pm, max_pm]. La ecuación (4.4) muestra el cálculo de la probabilidad para un ciclo $current\_cycle$ determinado.

$$pm = max\_pm - \frac{max\_pm - min\_pm}{max\_cycle} * current\_cycle \tag{4.4}$$

Este operador de mutación se diferencia de los demás en que la probabilidad de aplicación es alta en los ciclos iniciales del proceso de búsqueda, y va decrementándose a medida que transcurren las iteraciones. Esto ayuda a explorar grandes áreas al inicio del algoritmo, para luego estabilizar la búsqueda en un área más acotada, favoreciendo así la explotación.

Otra modificación introducida fue la división de la población en dos subpoblaciones que evolucionan simultáneamente pero de forma independiente. La justificación de esta decisión es favorecer la aceleración en el proceso de exploración del espacio de búsqueda ya que no sólo una población de individuos recorre la zona de soluciones, sino dos poblaciones más pequeñas. De esta forma, la posibilidad de que la población converja a un óptimo local se ve reducida a la mitad. Una pregunta obvia que puede surgir es ¿por qué dividir la población en dos y no en más grupos? La respuesta tiene fundamento en que esta versión de PSO trabaja adecuadamente con un cúmulo bastante reducido (10 partículas) por lo tanto cada subpoblación cuenta con sólo 5 partículas. Si se pretende utilizar vecindarios para aprovechar las bondades de ese modelo, una cantidad menor de partículas, podría resultar inapropiada. Todas las características de Bi-PSO antes mencionadas se siguen manteniendo pero se aplican a cada subpoblación de manera independiente en vez de a la población completa: vecindarios, modelos *lbest* y *gbest*, ecuaciones para actualizar velocidades y posiciones y, finalmente, el operador de mutación. La figura 4.3 ilustra esta mejora.
Nótese que la idea de utilizar varias poblaciones en simultáneo ha sido propuesto por varios autores [12, 32, 105, 182, 205] pero de forma diferente a la utilizada por Bi-PSO. En estos trabajos se emplean varios cúmulos de tamaño reducido pero que evolucionan compartiendo información relevante (los mejores) entre ellos. El intercambio de información facilita la búsqueda de soluciones ya que las mejores soluciones de cada subpoblación son utilizadas por los otros cúmulos. Además, en los trabajos previos, las subpoblaciones son dinámicas en el sentido que modifican su tamaño durante la exploración del espacio de búsqueda. El objetivo de este dinamismo es adaptar cada cúmulo dependiendo de la dificultad del problema que se resuelve. Los cúmulos dinámicos a menudo son re-armados durante el proceso evolutivo para que los individuos de cada grupo influencien a un conjunto diferente de partículas. En el caso de Bi-PSO las subpoblaciones evolucionan de forma totalmente independiente, es decir, sin compartir información. Las dos poblaciones que emplea Bi-PSO son de tamaño fijo (no cambia la cantidad de individuos de cada subpoblación durante la ejecución del algoritmo) y no se mueven partículas de una población a otra.

En el Algoritmo 4.1 se muestra un pseudo código del enfoque propuesto: Bi-PSO. En la fase de inicialización del cúmulo, a cada dimensión de las partículas se le asigna un valor dentro de un rango permitido, el cual depende exclusivamente del problema al que se aplica el algoritmo. Luego se inicializa cada dimensión del vector de velocidades en cero (líneas 1 a 7). En la fase de división, la población es separada en dos grupos y por cada uno de ellos se calculan los valores objetivos de cada partícula y los valores correspondientes a *pbest*, *lbest* y *gbest* (líneas 8 a 13). La evolución de cada grupo está regida por una cantidad máxima de ciclos (líneas 14 a 29). Por cada subpoblación se actualizan los líderes de cada vecindario, velocidad y posición correspondientes a cada partícula (líneas 17 a 22)

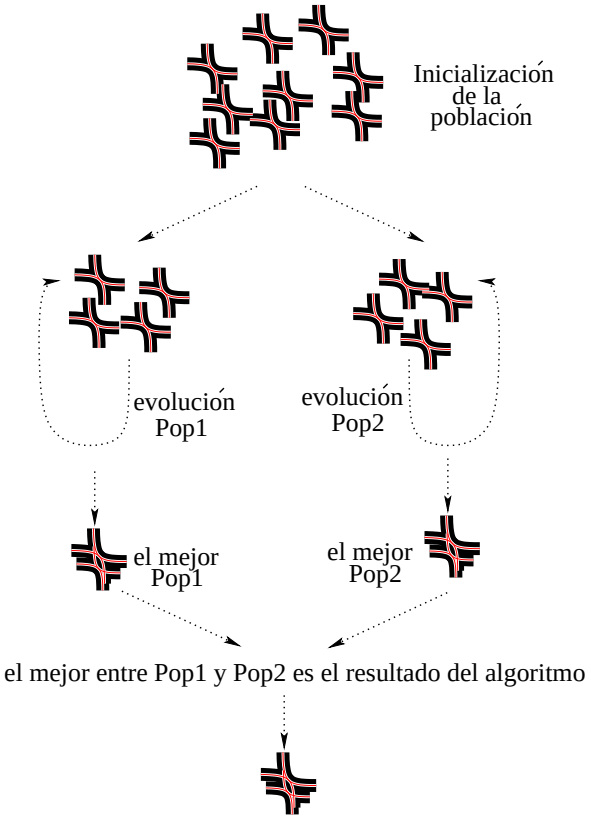


Figura 4.3: Proceso evolutivo de Bi-PSO.

para luego realizar una operación de control de rangos (*keeping*) en la cual se verifica que los rangos de cada dimensión de las partículas sean correctos (línea 23). Si alguna dimensión fuera incorrecta, se reinicializa dicha dimensión a un valor aleatorio dentro del rango permitido. Luego de actualizar la probabilidad de mutación dinámica se mutan las partículas si así lo determinara *pm* (líneas 24 y 25). Se calculan los valores objetivos de cada individuo, se actualizan los mejores de cada patícula y el mejor global de cada subgrupo (líneas 26 y 27). Cuando los ciclos de evolución finalizan, cada subpoblación posee un mejor valor encontrado. Se comparan esos valores para determinar el mejor de ellos y reportarlo como resultado del algoritmo (líneas 30 y 31).

4.4. Análisis de resultados

Para validar esta propuesta se utilizaron 13 funciones de minimización tomadas de [200]. Estas funciones constituyen un benchmark muy utilizado para validar algoritmos de optimización global ya que reúnen características deseables para verificar robustez y efectividad. El benchmark incluye funciones unimodales y multimodales de diferentes grados de complejidad, las cuales se describen en el Apéndice A. Todos los problemas son de minimización y cuentan con 30 variables de decisión. Las funciones unimodales f1 a f7 son relativamente sencillas de optimizar pero su complejidad aumenta cuando se incrementa

Algoritmo 4.1 Bi-PSO

1: **Fase de Inicialización del cúmulo:**
2: **Para cada** partícula i, $i \in [1, N]$ **hacer**
3: **Para cada** dimensión d, $d \in [1, D]$ **hacer**
4: Aplicar a x_{id} un valor aleatorio en el rango $[Xmin, Xmax]$
5: Aplicar a v_{id} el valor inicial cero
6: **Fin Para**
7: **Fin Para**
8: **Fase de División del cúmulo:**
9: **Para cada** sub-población **hacer**
10: Calcular el $fitness\_x_i$
11: Aplicar a *pbest* y *lbest* el correspondiente valor
12: Aplicar a *gbest* el mejor valor de la sub-población
13: **Fin Para**
14: **Fase de Búsqueda:**
15: **Mientras** $current\_cycle < max\_cycle$ **hacer**
16: **Para cada** sub-población **hacer**
17: **Para cada** partícula i, $i \in [1, N]$ **hacer**
18: Buscar el líder en el vecindario de x_i
19: Aplicar a *lbest* el mejor valor del vecindario
20: Calcular v_i con la ecuación (4.2)
21: Calcular x_{id} con la ecuación (4.3)
22: **Fin Para**
23: Controlar rangos
24: Actualizar pm
25: Mutar partículas dependiendo de pm
26: Calcular $fitness\_x_i$
27: Aplicar *pbest* y *gbest* el correspondiente valor
28: **Fin Para**
29: **Fin Mientras**
30: resultado = Mejor(mejorPop1,mejorPop2)
31: **Retornar** resultado

la dimensionalidad. La función f6 es discontinua con un único óptimo y f7 es una función con ruido que incorpora un valor aleatorio con distribución uniforme en el rango [0, 1]. Las funciones f8 a f13 cuentan con múltiples óptimos locales que se incrementan a medida que se aumenta la dimensionalidad de ellas. Estas últimas, principalmente, son las más complejas de resolver para muchos algoritmos de optimización.

En los experimentos se realizaron 50 ejecuciones independientes por función con un total de 120,000 evaluaciones de la función objetivo. Los parámetros que utilizó Bi-PSO se encuentran resumidos en la tabla 4.1 y la fundamentación de dichos valores se presenta en el Capítulo 7.

Los resultados de Bi-PSO fueron comparados con los de un algoritmo Evolución Diferencial (DE por sus siglas en inglés) propuesto por Velázquez [189]. DE es un algoritmo evolutivo específico para optimización de funciones cuyos espacios de búsqueda son conti-

nuos. El desempeño de los algoritmos basados en evolución diferencial ha sido sobresaliente en diversos estudios reportados en la literatura especializada [146]. El algoritmo DE realiza una serie de mutaciones basadas en la distribución de soluciones en la población de individuos, con lo cual la búsqueda es direccionada dependiendo de la ubicación de los mejores individuos. En [146, 189] pueden consultarse más detalles. También se utilizó en este estudio un algoritmo PSO básico que emplea una distribución de probabilidad exponencial en la generación de los coeficientes de pesos, particularmente en el factor de constricción. Este algoritmo, denominado PSO-E, fue propuesto por Krohling y Dos Santos Coelho [92] y las modificaciones incorporadas tienen el fin de evitar el estancamiento del método en óptimos locales. PSO-E fue evaluado en solamente 6 de las 13 funciones empleadas en este estudio.
Es importante mencionar que DE realizó, como Bi-PSO, la cantidad de 120,000 evaluaciones de la función objetivo por cada una de las 50 ejecuciones independientes que se efectuaron*. El algoritmo PSO-E se evaluó en 30 ejecuciones independientes con un total de 150,000 evaluaciones de la función objetivo por cada ejecución.
Los parámetros utilizados por DE y PSO-E se encuentran resumidos en la tabla 4.2.

Población	**ProbMutMin**	**ProbMutMax**	$\mathcal{C}_i$	$\mathcal{X}$	**Vecindario**	**ProbActGauss**
10	0.10	0.40	1.80	0.80	3	0.075

Tabla 4.1: Configuración de Bi-PSO para funciones sin restricciones.

Algoritmo	**Población**	$\mathcal{C}_i$	$\mathcal{X}$	**Cantidad descendientes** (λ)	**Coeficiente DE** (cf)
DE	30	-	-	150	0.9
PSO-E	30	2.05	0.729	-	-

Tabla 4.2: Parámetros correspondientes a DE y PSO-E.

La tabla 4.3 muestra los mejores valores obtenidos por cada algoritmo al finalizar las ejecuciones independientes. Nótese que se encuentran remarcados en negritas los valores diferentes a los óptimos ya que en general los algoritmos encuentran los mejores conocidos. Como puede observarse en la tabla, el desempeño de Bi-PSO y DE es bastante similar ya que tanto Bi-PSO como DE encuentran el óptimo (o mejor valor conocido) en las funciones f1, f2, f6, f7, f8, f9, f10, f11, f12 y f13. Para f3 y f4 Bi-PSO aproxima el óptimo pero no lo encuentra como DE. Para f5 ni Bi-PSO ni DE encuentran el óptimo pero Bi-PSO halla un valor bastante más cercano al benchmark que DE. Con respecto a los valores obtenidos con PSO-E se puede apreciar que sólo para f11, f12 y f13 los resultados son comparables con Bi-PSO y DE ya que encuentra el óptimo. Para f10 el mejor valor de PSO-E se acerca bastante al óptimo conocido, no siendo así para f8 y f9.

*Se agradece la buena predisposición del Mg. Jesús Velázquez Reyes y el Dr. Carlos Coello Coello en proveer el código fuente de DE con el cual se efectuaron las pruebas para la comparación directa del desempeño de los algoritmos.

Tabla 4.3: Mejores valores obtenidos con Bi-PSO, DE y PSO-E.

Función	**Benchmark**	**Bi-PSO**	**DE**	**PSO-E**
f1	0.00000	0.00000	0.00000	-
f2	0.00000	0.00000	0.00000	-
f3	0.00000	**3.50228**	0.00000	-
f4	0.00000	**0.11688**	0.00000	-
f5	0.00000	0.00010	**0.04031**	-
f6	0.00000	0.00000	0.00000	-
f7	0.00000	0.00000	0.00000	-
f8	-12,569.48661	-12,569.48661	-12,569.48661	**-11,262.00000**
f9	0.00000	0.00000	0.00000	**12.93450**
f10	0.00000	0.00000	0.00000	**0.00830**
f11	0.00000	0.00000	0.00000	0.00000
f12	0.00000	0.00000	0.00000	0.00000
f13	0.00000	0.00000	0.00000	0.00000

La tabla 4.4 muestra los valores medios sobre el total de ejecuciones. Como puede observarse, para f3, f4, f5 y f9 los valores medios de DE son menores que los obtenidos por Bi-PSO. Sólo para f8, Bi-PSO alcanza un valor medio inferior que el obtenido con DE. Los valores medios de PSO-E se alejan bastante de los obtenidos por Bi-PSO y DE posiblemente porque realiza 20 ejecuciones menos que los dos últimos algoritmos.

Tabla 4.4: Valores medios obtenidos con Bi-PSO, DE y PSO-E.

Función.	**Benchmark**	**Bi-PSO**	**DE**	**PSO-E**
f1	0.00000	0.00000	0.00000	-
f2	0.00000	0.00000	0.00000	-
f3	0.00000	178.54809	0.00000	-
f4	0.00000	0.77809	0.00000	-
f5	0.00000	28.28478	2.71842	-
f6	0.00000	0.00000	0.00000	-
f7	0.00000	0.00000	0.00000	-
f8	-12,569.48661	-12,569.48632	-12,550.53648	-10,360.00000
f9	0.00000	0.84162	0.23879	30.87920
f10	0.00000	0.00000	0.00000	0.00830
f11	0.00000	0.00000	0.00000	0.03490
f12	0.00000	0.00000	0.00000	0.21270
f13	0.00000	0.00000	0.00000	0.11950

Los peores valores obtenidos por cada algoritmo sobre el total de ejecuciones pueden ser observados en la tabla 4.5. Para f3, f4, f5 y f9 los peores valores encontrados por Bi-PSO se alejan considerablemente del óptimo, no siendo así con los encontrados por DE.

Tabla 4.5: Peores valores obtenidos con Bi-PSO, DE y PSO-E.

Función.	**Benchmark**	**Bi-PSO**	**DE**	**PSO-E**
f1	0.00000	0.00000	0.00000	-
f2	0.00000	0.00000	0.00000	-
f3	0.00000	420.03210	0.00000	-
f4	0.00000	1.72454	0.00000	-
f5	0.00000	80.14255	10.43110	-
f6	0.00000	0.00000	0.00000	-
f7	0.00000	0.00000	0.00000	-
f8	-12,569.48661	-12,569.48572	-12,545.60177	-9,112.00000
f9	0.00000	1.37462	0.43138	74,62150
f10	0.00000	0.00000	0.00000	0.00830
f11	0.00000	0.00000	0.00000	0.40510
f12	0.00000	0.00000	0.00000	1.04470
f13	0.00000	0.00000	0.00000	0.79760

Para f8 el peor valor obtenido por DE está a mayor distancia del óptimo que el hallado por Bi-PSO. Los peores valores obtenidos con PSO-E se alejan mucho más del óptimo que los encontrados con Bi-PSO lo cual indica que Bi-PSO es más robusto que PSO-E.

Como conclusión de las tablas 4.4 y 4.5 puede afirmarse que la robustez de Bi-PSO es levemente inferior que la de DE ya que las pequeñas desviaciones en los valores medios y peores de cada algoritmo con respecto a los óptimos, son más notorias en el primero. No obstante, esto sucede para sólo 4 funciones de las 13 ya que para las demás Bi-PSO encuentra los óptimos conocidos o valores más cerca que DE. Con respecto a PSO-E, el algoritmo propuesto Bi-PSO demuestra más robustez, además de tener mejor desempeño que el otro.

Las figuras 4.4 y 4.5 ilustran los errores de las medias y peores con respecto al benchmark. En estas gráficas puede obsevarse cómo Bi-PSO posee errores destacados en las funciones f3 y f5 y PSO-E en las funciones f8 y f9. Excepto por f8, DE evidencia más robustez que las dos variantes de PSO analizadas.

Para inferir los resultados de las pruebas de desempeño, la observación directa de los mejores, medias y peores no constituye una prueba suficiente. Para tal fin, el análisis estadístico de confiabilidad aporta una prueba más contundente.

4.5. Estudio estadístico de resultados

El estudio estadístico se realizó mediante el análisis de varianza entre DE y Bi-PSO utilizando los mejores valores de las 50 ejecuciones independientes de cada uno. No se

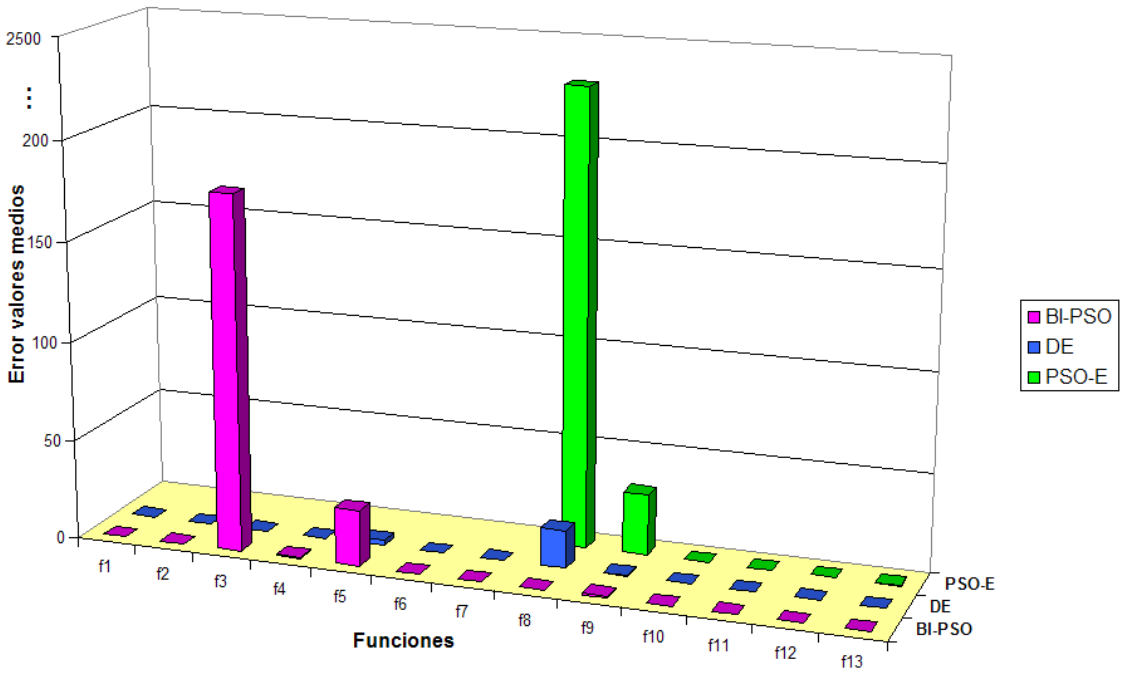


Figura 4.4: Error de valores medios.

incluye en este análisis el algoritmo PSO-E ya que no se cuenta con los valores necesarios para cada una de las ejecuciones.
Se aplicó el análisis no paramétrico Kruskal-Wallis [71] debido a que los dos conjuntos de 50 valores (muestras) no poseen una distribución normalmente distribuida. Esto último fue determinado con el test de Kolmogorov-Smirnov [21].
El test de Kruskal-Wallis arroja un valor denominado p por cada función, el cual determina el rechazo o no de la hipótesis nula: "no existe diferencia significativa entre los algoritmos". Si el p es cero o un valor cercano a cero (en general se utiliza un umbral de 0.05) el test sugiere que al menos un par de valores en la muestra son significativamente diferentes es decir, las muestras son estadísticamente significativas. De otra manera, nada puede inferirse sobre las muestras.
La tabla 4.6 muestra los valores de p para cada una de las funciones. Los resultados indican que los valores obtenidos por Bi-PSO para todas las funciones excepto f3, f4, f5, f8 y f9 no son estadísticamente diferentes, con lo cual los algoritmos presentan un comportamiento similar en cuanto a la obtención de los óptimos y la robustez.
Como indican los valores p para f3, f4, f5, f8 y f9, los de DE son estadísticamente diferentes de los obtenidos con Bi-PSO. Si se observan los mejores valores alcanzados por DE (tabla 4.3), éstos superan a los de Bi-PSO. Para f5 el mejor valor obtenido por Bi-PSO, supera al obtenido por DE con lo cual la diferencia significativa favorece a Bi-PSO para esta función. Si se observan en la tabla 4.3 los mejores valores obtenidos para f8 y f9, éstos son similares para ambos algoritmos con lo cual la confiabilidad estadística podría

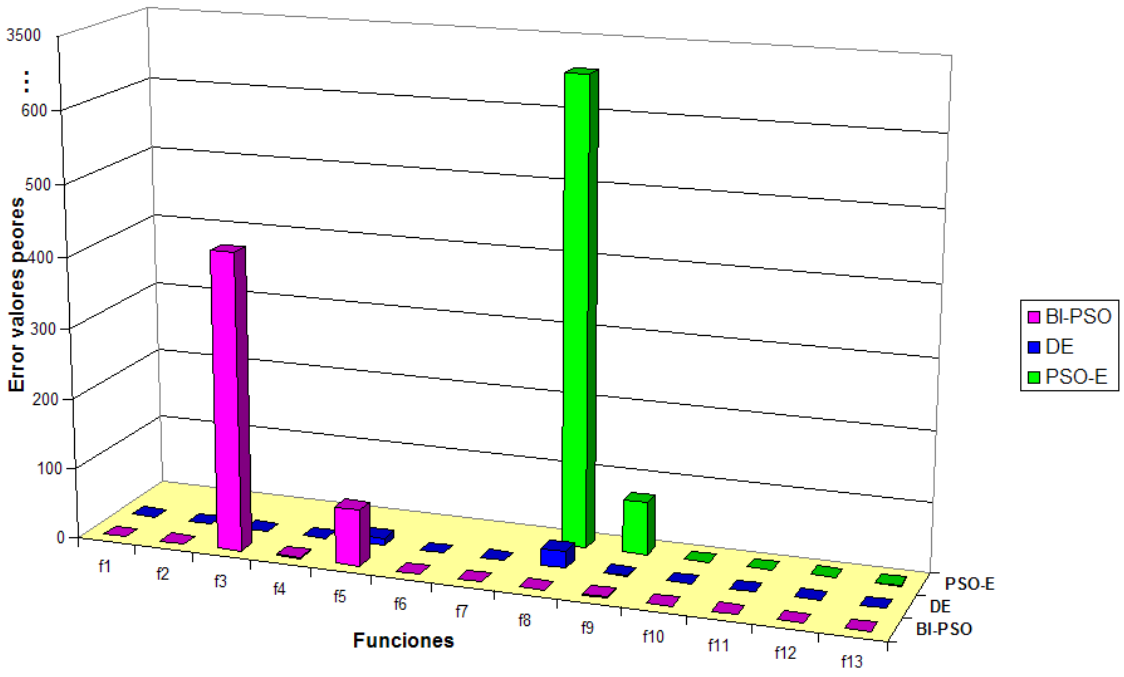


Figura 4.5: Error de valores peores.

Tabla 4.6: Test de Kruskal-Wallis para DE y Bi-PSO.

Función	p	LI	EV	LS
f1	1.0000	-	-	-
f2	1.0000	-	-	-
f3	0.0000	-61.3721	-50.0000	-38.6279
f4	0.0000	-0.8797	-0.7780	-0.6763
f5	6.6437e-011	-49.2521	-37.8800	-26.5079
f6	1.0000	-	-	-
f7	1.0000	-	-	-
f8	8.9338e-010	-44.8755	-34.0000	-23.1245
f9	2.0054e-009	-44.1559	-33.2800	-22.4041
f10	1.0000	-	-	-
f11	0.3162	-	-	-
f12	1.0000	-	-	-
f13	1.0000	-	-	-

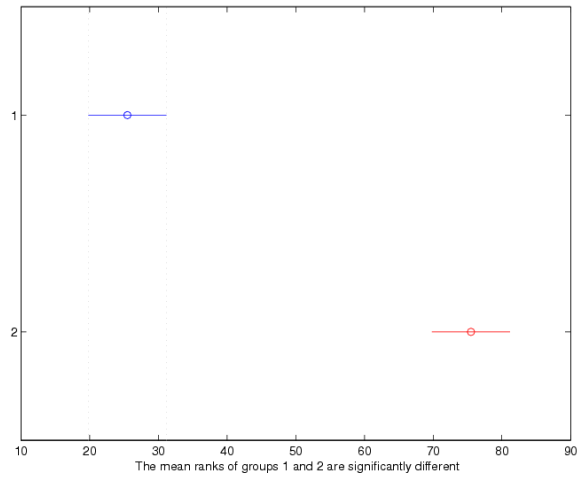
indicar diferencias en cuanto a la robustez de los mismos (observar las tablas 4.4 y 4.5).

Para las funciones en las que se halló diferencia significativa, se aplicó el test de Tukey [112] para determinar bajo qué condiciones estas diferencias son significativas. Este

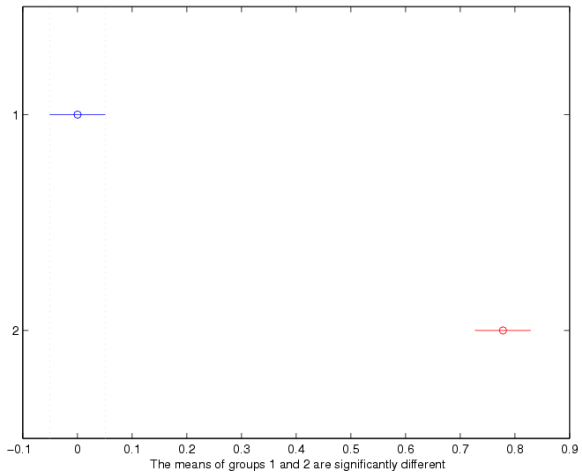
test arroja tres valores los cuales representan un valor estimado (EV) en un rango compuesto por un límite inferior (LI) y un límte superior (LS). Si el valor estimado se encuentra en un rango que no incluye el valor cero, esto significa que los resultados del test de Kruskal-Wallis se confirman, es decir, son significativamente diferentes. Como puede observarse en la tabla 4.6 ninguno de los rangos ([LI, LS]) posee el valor cero con lo cual las diferencias estadísticas son significativas. Esta conclusión se repite en cada una de las gráficas de la figura 4.6 en donde 1- hace referencia a DE y 2- hace referencia a Bi-PSO. Cuando las diferencias no son significativas las barras correspondientes a cada muestra se solapan, lo cual no ocurre en este caso.

4.6. Conclusiones

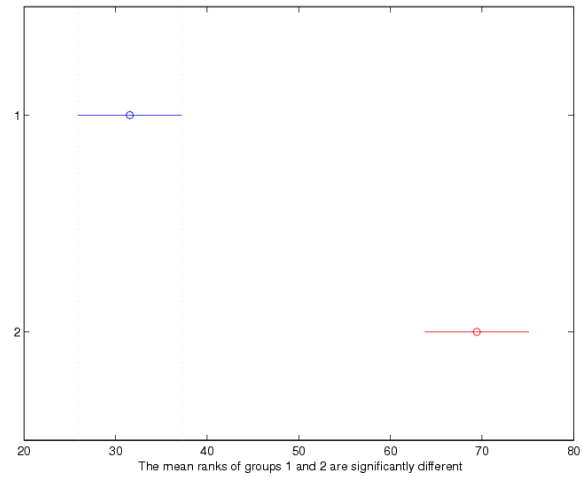
En este capítulo se introdujo un algoritmo basado en PSO para optimizar problemas mono-objetivo sin restricciones. Los resultados de Bi-PSO mostraron ser superiores a los obtenidos con otro algoritmo basado en PSO (PSO-E) y, buenos y competitivos con respecto a los obtenidos por un algoritmo basado en Evolución Diferencial (DE). Esto último se corroboró tanto en la comparación directa, como en el análisis estadístico de confiabilidad realizado posteriormente. Bi-PSO, a pesar de ser un algoritmo más simple que DE (que utiliza un complejo procedimiento de cómputo de soluciones), tuvo un desempeño similar a este último. Aunque se detectaron algunas pequeñas desviaciones en algunos valores de Bi-PSO, en general, la calidad de resultados fue buena, si bien resulta evidente que BI-PSO es menos robusto que DE como lo demuestran los valores medios y peores. Con respecto a PSO-E, ambos algoritmos resultaron ser poco robustos en al menos dos funciones.



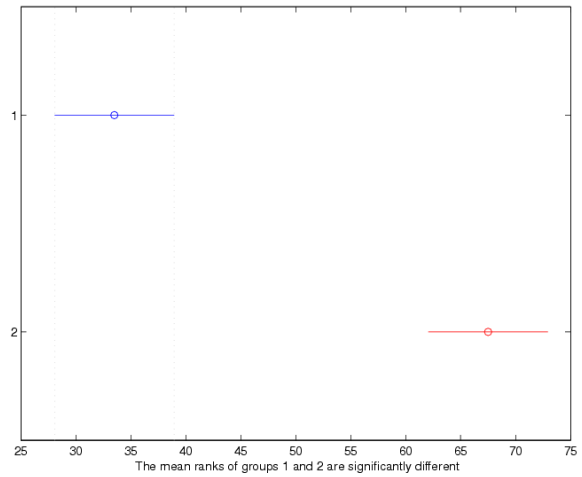
(a) f3



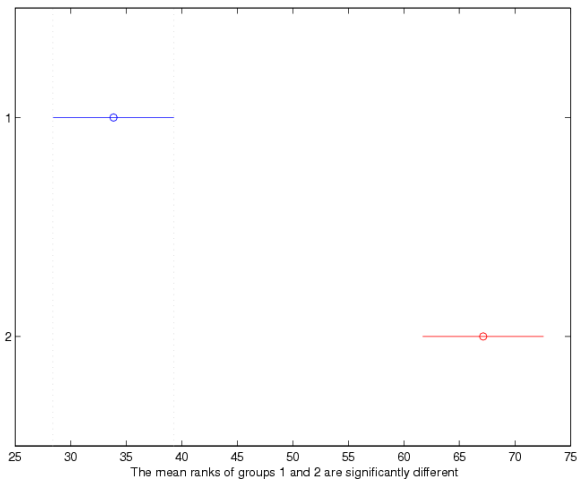
(b) f4



(c) f5



(d) f8



(e) f9

Figura 4.6: Test de Tukey para DE (1-) y Bi-PSO (2-).

Capítulo 5

Solución de Problemas Mono-objetivo con restricciones

El proceso de optimización de problemas que involucran funciones con restricciones es un tópico fundamental de estudio, ya que la mayoría de problemas del mundo real presentan restricciones de algún tipo. Aunque existen muchos métodos para resolver este tipo de problemas de optimización, es muy difícil encontrar uno que sea siempre eficiente para cualquier tipo de restricciones. Existen diversos esquemas de manejo de restricciones. En este capítulo, se describen los métodos que han sido utilizados con más frecuencia en la literatura especializada. Posteriormente, se presenta un algoritmo PSO para resolver problemas de optimización con restricciones. La propuesta es evaluada con un compendio de casos de prueba que incluye: un benchmark de 24 funciones, así como problemas de diseño óptimo de armaduras, funciones de diseño ingenieril de piezas y un problema de despacho de cargas eléctricas.

Contenido del Capítulo

5.1. Organización del capítulo

Este capítulo comienza con una breve introducción a la optimización de funciones con restricciones. A continuación se describe una taxonomía que clasifica las técnicas de manejo de restricciones en grupos, dependiendo de las características del método. Luego se presenta una versión del algoritmo PSO para resolver problemas con restricciones, llamada **CPSO-shake**. El algoritmo fue validado con 4 casos de prueba diferentes: 24 funciones de un benchmark muy utilizado en optimización, 3 funciones de diseño óptimo de armaduras, 4 problemas de diseño ingenieril de piezas y un problema de despacho de cargas eléctricas. Estos tres últimos casos son problemas de la vida real. El capítulo finaliza con las conclusiones del comportamiento de CPSO-*shake* para problemas de optimización con restricciones.

5.2. Optimización con restricciones

Muchos problemas del mundo real deben ser resueltos considerando un conjunto de restricciones. Esas restricciones limitan el espacio de búsqueda de soluciones a una zona especial que se denomina factible. Dependiendo de las restricciones, la zona factible puede ser única o estar compuesta de varias zonas dispersas en el espacio de búsqueda, lo cual dificulta aún más la solución del problema. Contrario a lo que pudiera pensarse, el uso de restricciones, si bien reduce la región del espacio de búsqueda donde residen las soluciones de interés, no necesariamente vuelve al problema más fácil de resolver. De hecho, lo suele hacer más difícil debido a que se complica el proceso de alcanzar la solución de interés (o sea, el óptimo global). Esto sucede como consecuencia de las irregularidades que las restricciones introducen en el espacio de búsqueda.

5.3. Manejo de restricciones

Diferentes mecanismos que permiten trabajar con restricciones de igualdad, desigualdad, lineales y no lineales han surgido en la última década pero la selección del mecanismo adecuado para un problema específico continúa siendo un problema abierto. Normalmente, las técnicas de manejo de restricciones agregan más parámetros al método de búsqueda y/o aumentan su costo computacional [27].
Una posible categorización de las técnicas de manejo de restricciones es la siguiente [25, 91, 122]:

- **Rechazo de soluciones infactibles:** todas las soluciones que se detectan como no factibles, se descartan como soluciones o simplemente son ignoradas.

- **Funciones de penalización:** es la técnica más popular de todas. Consiste en castigar o penalizar a las soluciones no factibles disminuyendo su aptitud, de manera que ellas no sean favorecidas en el proceso de selección. La utilización de esta técnica tiene una desventaja importante: el usuario debe establecer los factores de penalización y muchas veces esto debe hacerse para cada problema en forma particular.

- **Preservación de factibilidad:** presupone que inicialmente (en el proceso de búsqueda) todas las soluciones se encuentran en el espacio factible. Mediante el uso de operadores especiales, las soluciones son transformadas a nuevas soluciones factibles.
- **Métodos de Pareto:** utilizan los conceptos de optimización multiobjetivo como la no dominancia de las soluciones para ordenar las soluciones dependiendo del grado de violación que posea cada una de ellas.
- **Algoritmos de reparación:** mediante la utilización de operadores especiales convierte soluciones no factibles en factibles. La versión de la solución reparada puede, dependiendo de un valor de probabilidad, reemplazar a la original. Estos algoritmos se utilizan frecuentemente en optimización combinatoria.

Otras técnicas son [27]:

- **Operadores y representaciones especiales:** utilizados principalmente en problemas donde la localización de una solución factible es extremadamente difícil y se requiere una codificación de soluciones que facilite la búsqueda. Ejemplo de esto son las aplicaciones de Davis [34], las llaves aleatorias [9], GENOCOP [124] y los mapas homomorfos [91].
- **Separación de objetivos y restricciones:** las restricciones y la función objetivo son manejadas de forma independiente. De esta manera no se combina el grado de violación de las restricciones con el valor de la función que se optimiza.
- **Métodos híbridos:** son técnicas que se combinan con otras que manejan restricciones. Por ejemplo, existen métodos que utilizan la minimización de Lagrange combinada con funciones de penalización [2].

5.4. El algoritmo propuesto: CPSO-shake

CPSO-shake se basa en el algoritmo que se presentó en el capítulo anterior, para optimización sin restricciones, al cual se le ha agregado un método de manejo de restricciones. Comparte con Bi-PSO la ecuación (4.2) de actualización de la velocidad, la ecuación (4.3) de actualización de las partículas, la topología circular de vecindarios, las dos poblaciones evolucionando en paralelo y el operador de mutación dinámica.

El mecanismo de manejo de restricciones es uno de los más simples posibles de implementar. Se rige por la siguiente regla: "una partícula factible siempre es preferida sobre una no factible". Así, cuando dos partículas son comparadas, se verifica su factibilidad. Si ambas son factibles, se elige aquélla con mejor valor objetivo. Si ambas son no factibles, se elige aquélla que se encuentre más cerca de la región factible. Para hacer esto, el algoritmo almacena la violación más grande obtenida por cada restricción, en cada generación del proceso de búsqueda. Cuando un individuo es detectado como infactible, la suma de

las violaciones (normalizadas con respecto al mayor valor obtenido hasta el momento) es considerada como su distancia a la región factible. La figura 5.1 ilustra el mecanismo de selección de una partícula cuando se está realizando el manejo de restricciones propuesto. Se considera un problema de minimización de una función que posee dos restricciones. En la figura, para el caso A, el líder será P1 por encontrarse en la zona factible de las dos restricciones mientras que para el caso B, la partícula seleccionada será P2 porque la distancia a la zona factible es menor que si se eligiera P1. Finalmente, para el caso C, el líder es P1 porque ambas partículas son factibles pero P1 posee el menor valor de función objetivo. Este esquema es utilizado por CPSO-shake cuando seleccionan las partículas *pbest*, *gbest* y *lbest*.

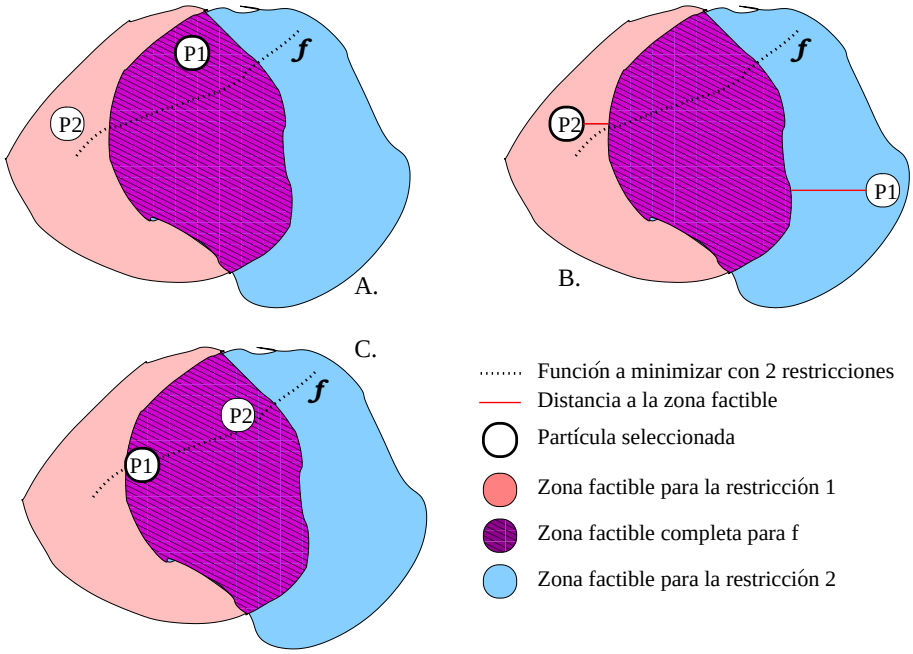


Figura 5.1: Selección de líderes en CPSO-shake.

Es una práctica común en optimización con restricciones, transformar las restricciones de igualdad en desigualdades, debido a la dificultad inherente en tratar ese tipo de restricciones. De hecho, la mayoría de las técnicas de manejo de restricciones utilizadas en los algoritmos evolutivos realizan esta conversión para poder resolver adecuadamente problemas que poseen este tipo de restricciones. La transformación se define en la ecuación (5.1).

$$|h_e(\vec{x})| - \epsilon \leq 0 \tag{5.1}$$

donde h_e es el valor de la restricción de igualdad, ϵ es un factor de tolerancia que normalmente se fija en el valor 0.0001 [160]. CPSO-shake trata las restricciones de igualdad empleando esta transformación, pero sin fijar el valor del factor de tolerancia. Es decir, CPSO-shake utiliza un factor de tolerancia dinámico que es adaptado tres veces en el total de ciclos de evolución. Cuando el proceso de búsqueda comienza, el factor de tolerancia

toma un valor inicial de 0.1. A medida que los ciclos del algoritmo se van sucediendo, estos se van contabilizando para realizar en total otras tres actualizaciones del factor, tomando ϵ los valores 0.1, 0.01, 0.001 y 0.0001 durante una ejecución del algoritmo. Asumiendo que el algoritmo será ejecutado durante Q ciclos, el valor del factor de tolerancia cambiará acorde al esquema mostrado en la figura 5.2.

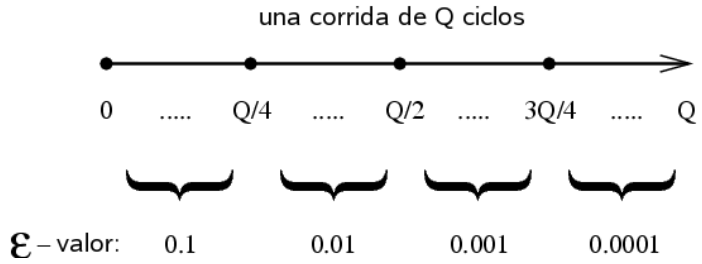


Figura 5.2: Variación de ϵ durante una ejecución de CPSO-shake.

La principal ventaja en la utilización del factor de tolerancia dinámico es favorecer la existencia de soluciones factibles al comienzo de la búsqueda, permitiendo que soluciones "casi" factibles sean tratadas como factibles. A medida que se incremente el número de ciclos y se decremente ϵ, CPSO-shake comenzará a converger a soluciones que satisfagan las restricciones de igualdad con una mayor certeza.
El proceso de adaptación del factor de tolerancia fue realizado sólo tres veces con el objetivo de que el valor final sea 0.0001, además de que el tiempo en el que el algoritmo utilice dicho valor sea considerable: 1/4 del total de las iteraciones de CPSO-shake. Un valor inferior a esa cantidad podría no proveer suficiente flexibilidad para la relajación de las restricciones, y un valor mayor podría no permitir encontrar soluciones en la región factible.

En algunos casos de optimización de funciones difíciles, se detectaron problemas de estancamiento en zonas del espacio de búsqueda no cercanas al óptimo [17]. Una forma de tratar este problema es forzar el movimiento de las partículas, fuera de las zonas de estancamiento. Para tal fin se incorpora en CPSO-shake un mecanismo denominado *shake* (sacudida) que redirecciona a los individuos a la zona de una partícula *pbest*. Este mecanismo es aplicado cuando el porcentaje de individuos infactibles es mayor al 10 %. Este valor fue fijado para que en la población siempre existan individuos infactibles ya que no es conveniente, por cuestiones de diversidad, mantener sólo soluciones factibles. Además, la conservación de partículas infactibles colabora en el proceso de búsqueda cuando se trata de resolver problemas con restricciones activas, ya que permiten explorar los límites de las regiones factibles-infactibles.
Para implementar el mecanismo de sacudida (*shake*), se mueven las partículas a otras zonas del espacio de búsqueda. Aunque se podrían realizar movimientos en direcciones aleatorias, se considera mejor guiarlas hacia zonas prometedoras. Por esta razón, se selecciona aleatoriamente una partícula *pbest* y se aplica el mecanismo, ejecutando la ecuación (5.2).

$$v_{id} = \mathcal{X} v_{id} + c_1 r_1 (pbest_{SEL_d}) \tag{5.2}$$

donde v_{id} es la dimensión d de la velocidad de la $i-$ésima partícula, la cual fue seleccionada para que se le aplique el mecanismo *shake*, $\mathcal{X}$ es el factor de constricción, c_1 es el factor de aprendizaje personal, el cual es multiplicado por un número aleatorio $r_1 \in [0, 1]$ y $pbest_{SEL_d}$ es la dimensión d de la partícula *pbest* seleccionada aleatoriamente. El mecanismo no se aplica a toda la población, sino sólo al 50 % de las partículas, para evitar una presión excesiva hacia la ubicación de las partículas *pbest*. Se selecciona una partícula $pbest_{SEL}$ diferente en cada ciclo de vuelo.

El Algoritmo 5.1 muestra el pseudo código de CPSO-shake. En la fase de inicialización del cúmulo a cada dimensión de las partículas se les asigna un valor dentro del rango correspondiente a la definición del problema a resolver. El vector de velocidades asociado a cada partícula es inicializado con el valor cero (líneas 1 a 7). En la fase de división, la población es dividida en dos grupos y por cada uno de ellos se calcula el valor objetivo, el valor *pbest* y *lbest* de cada partícula. También se calcula el valor *gbest* que influenciará cada subpoblación (líneas 8 a 13). La evolución de cada subpoblación comienza en la línea 14, y se ejecuta una cantidad fija de ciclos (hasta línea 34). Por cada subpoblación se actualizan los líderes de cada vecindario (*lbest*), se actualizan las velocidades y las posiciones de las partículas (líneas 15 a 22). En la línea 23 se realiza una operación de control de rangos (*keeping*) en la cual se verifica que los valores de cada dimensión de las partículas sean correctos. Si alguna dimensión tuviera un valor fuera del rango establecido, se reinicializa dicha dimensión con un valor aleatorio dentro del rango permitido. Luego de la selección aleatoria de la partícula $pbest_{SEL}$ (línea 24) se controla el porcentaje de individuos infactibles para verificar si el mecanismo de sacudida (*shake*) debe ser aplicado. Si así fuera, entonces se aplica el mecanismo al 50 % de la población (líneas 25 a 27). Se actualiza la probabilidad de mutación dinámica y se mutan las partículas si así lo determinara *pm* (líneas 28 y 29). Se calculan los valores de las funciones objetivo de cada partícula, se actualizan los mejores de cada partícula y el mejor global de cada subgrupo (líneas 30 y 31). En la línea 32 se actualiza el factor de tolerancia si correspondiera. Cuando los ciclos de evolución finalizan, cada subpoblación posee el mejor valor que ha encontrado. Se comparan esos valores y, el mejor de ellos, se reporta como resultado del algoritmo (líneas 35 y 36).

A continuación se presentan los cuatro casos de prueba con los que se validó la propuesta.

5.5. Benchmark de 24 Funciones Clásicas

Se utilizaron 24 funciones de prueba de minimización con restricciones, propuestas en [104], las cuales constituyen un benchmark empleado para validar algoritmos de optimización con restricciones. La descripción de cada una de las funciones puede ser encontrada

Algoritmo 5.1 CPSO-shake

Fase de Inicialización del cúmulo:
Para cada partícula i, $i \in [1, N]$ **hacer**
  Para cada dimensión d, $d \in [1, D]$ **hacer**
    Aplicar a x_{id} un valor aleatorio en el rango $[Xmin, Xmax]$
    Aplicar a v_{id} el valor inicial cero
  Fin Para
Fin Para
Fase de División del cúmulo:
Para cada sub-población **hacer**
  Calcular el $fitness\_x_i$
  Aplicar a *pbest* y *lbest* el correspondiente valor
  Aplicar a *gbest* el mejor valor de la sub-población
Fin Para
Fase de Búsqueda:
Mientras $current\_cycle < max\_cycle$ **hacer**
  Para cada sub-población **hacer**
    Para cada partícula i, $i \in [1, N]$ **hacer**
      Buscar el líder en el vecindario de x_i
      Aplicar a *lbest* el mejor valor del vecindario
      Calcular v_i con la ecuación (4.2)
      Calcular x_{id} con la ecuación (4.3)
    Fin Para
    Controlar rangos
    Seleccionar $pbest_{SEL}$
    Si porcentaje de infactibles > 10 **luego**
      Aplicar mecanismo de *shake* (ec. (5.2)) al 50 % de la población
    Fin Si
    Actualizar pm
    Mutar partículas dependiendo de pm
    Calcular $fitness\_x_i$
    Aplicar a *pbest* y *gbest* su correspondiente valor
    Actualizar factor de tolerancia si corresponde
  Fin Para
Fin Mientras
resultado = Mejor(mejorPop1,mejorPop2)
Retornar resultado

en el Apéndice B. Todas las funciones fueron trasformadas a problemas de minimización y todas las restricciones de igualdad fueron transformadas a desigualdades utilizando el factor de tolerancia ϵ.
Para comparar el desempeño de CPSO-shake se utilizaron dos algoritmos representativos del estado del arte en optimización con restricciones: Dynamic Multi-swarm Particle Swarm Optimizer [104] (DMS-C-PSO) y Approximation Evolution Strategy with Sto-

Tabla 5.1: Configuración de CPSO-shake para funciones con restricciones.

Población	**ProbMutMin**	**ProbMutMax**	$\mathcal{C}_i$	$\mathcal{X}$	**Vecindario**	**ProbActGauss**
10	0.10	0.40	1.80	0.80	3	0.075

Tabla 5.2: Configuración de DMS-C-PSO y AESSR.

Algoritmo	**Población**	$\mathcal{C}_i$	w	$Vmax$	**Descen.**(μ)	**Escalar**(γ)
DMS-C-PSO	60*	1.49445	0.729	$0.5\times(Xmax - Xmin)$	-	-
AESSR	400	-	-	-	60	0.85

* (3 partículas $\times$20 cúmulos).

chastic Ranking [159] (AESSR). La comparación con DMS-C-PSO* es directa ya que se dispone del código fuente y esto permite realizar las ejecuciones, mientras que con AESSR la comparación es indirecta ya que sólo se dispone de los resultados publicados.
DMS-C-PSO es un algoritmo basado en PSO que incluye un método de manejo de restricciones basado en sub-poblaciones (*sub-swarms*), las cuales se asignan dependiendo de la dificultad de las restricciones. Además, cuenta con varios cúmulos (*multi-swarms*) dinámicos que se combinan con un método de búsqueda local para mejorar la búsqueda de soluciones. Las sub-poblaciones se reagrupan cada cierta cantidad fija de iteraciones. Más detalles pueden encontrarse en [104].
AESSR es una aproximación de una estrategia evolutiva cuya meta es la reducción de evaluaciones de la función objetivo. Utiliza dos modelos de optimización para llevar a cabo dicha reducción: uno que evalúa la función objetivo y otro que evalúa una función de penalización sobre las restricciones que no se satisfacen. Para una descripción más detallada ver [159].

Se realizaron 25 ejecuciones independientes con 350,000 evaluaciones de la función objetivo por cada ejecución de los algoritmos CPSO-shake y DMS-C-PSO. El algoritmo AESSR realizó 500,000 evaluaciones de la función objetivo. La cantidad de evaluaciones para CPSO-shake fue fijada en dicho valor porque, con un número mayor de evaluaciones, la mejora de los resultados no era significativa (determinado en un análisis empírico). Los parámetros que utilizó CPSO-shake se encuentran resumidos en la tabla 5.1 y la justificación de los valores seleccionados se encuentra en el Capítulo 7. Los parámetros de DMS-C-PSO y AESSR se muestran en la tabla 5.2.

La tabla 5.3 muestra los mejores valores encontrados con cada algoritmo. Todas las soluciones son factibles, excepto las que se indican con (*). Como se puede observar,

*Se agradece la buena predisposición del Profesor P. N. Suganthan en proveer el código fuente de DMS-C-PSO con el cual se efectuaron las pruebas para la comparación directa del desempeño de los algoritmos.

CPSO-shake, con un menor número de evaluaciones, pudo alcanzar los valores óptimos (o mejores conocidos) al igual que AESSR. De hecho, para las funciones g2, g8, g10, g11 y g17, CPSO-shake supera en desempeño a AESSR. Por el contrario, AESSR supera a CPSO-shake en las funciones g4, g7, g19 y g23. Para las funciones g20, g21 y g22, ambos algoritmos no obtuvieron soluciones factibles. Con respecto a DMS-C-PSO los resultados son contundentes, CPSO-shake encontró mejores valores para las funciones g1, g3, g4, g5, g6, g7, g9, g10, g13, g16, g17, g18 y g19.

Las desviaciones de los valores medios (sobre el total de ejecuciones), con respecto a los valores de referencia (benchmark) pueden ser observados en la tabla 5.4. En general, para algunas funciones, los errores obtenidos con CPSO-shake son levemente superiores a los de AESSR (g7, g9, g13, g14, g15, g16, g18) y grandes en otros (g4, g5, g6, g10, g17, g19). AESSR obtuvo peores valores de error para las funciones g2, g8, g11 y g23. Los errores obtenidos por DMS-C-PSO son peores en la mayoría de los casos que los de CPSO-shake y AESSR, con una diferencia pronunciada.

Los valores de error para los peores valores obtenidos por los algoritmos en las 25 ejecuciones con respecto al benchmark, se muestran en la tabla 5.5. Para la mayoría de las funciones, los peores valores alcanzados por CPSO-shake son mayores que los que hallan AESSR y DMS-C-PSO. Este hecho refleja una variabilidad importante de las soluciones como consecuencia de que el algoritmo obtiene el óptimo (o un valor cercano a él) en algunas ejecuciones pero en otras los mejores valores se alejan bastante del benchmark.
La figura 5.3 ilustra los errores de las medias con respecto al valor óptimo. En la gráfica no se muestran los errores de las funciones g20, g21, g22 y g23 debido a que los algoritmos no encuentran soluciones factibles. Visiblemente DMS-C-PSO es el algoritmo con mayores desviaciones de medias para al menos 4 funciones: g4, g5, g6, g10 y en menor escala para g16. Las desviaciones de los valores medios obtenidas con CPSO-shake son elevadas, aunque menores que las de DMS-C-PSO, para las funciones g5, g6, g10 y, en una escala bastante menor, para g4, g17 y g19. Visualmente las desviaciones de los valores medios obtenidos con AESSR son prácticamente imperceptibles. La figura 5.4 grafica las desviaciones de los peores valores encontrados con cada algoritmo con respecto al benchmark. CPSO-shake es el que posee errores más elevados en las funciones más complejas, es decir, en: g4, g5, g6, g9, g10. En una escala menor en las funciones g14, g15 y g17. En estas tres últimas funciones DMS-C-PSO, a pesar de obtener errores más grandes en los valores medios, muestra desviaciones menores de las peores soluciones halladas.
A partir de estos resultados, es claro que la robustez de CPSO-shake es un aspecto que debe ser mejorado cuando se consideran funciones difíciles, ya que en las menos complejas, la robustez es adecuada.

Se realizó un estudio estadístico de desempeño entre CPSO-shake y DMS-C-PSO. No se pudo incorporar al estudio el algoritmo AESSR debido a que no se cuenta con los valores obtenidos en cada una de las 25 ejecuciones independientes.
Se efectuó un análisis de varianza con el test no paramétrico de Kruskal-Wallis debido a que no se cumple con el supuesto de normalidad de las muestras (hecho verificado con el test de Kolmogorov-Smirnov). Los valores p obtenidos se encuentran en la tabla 5.6. En la

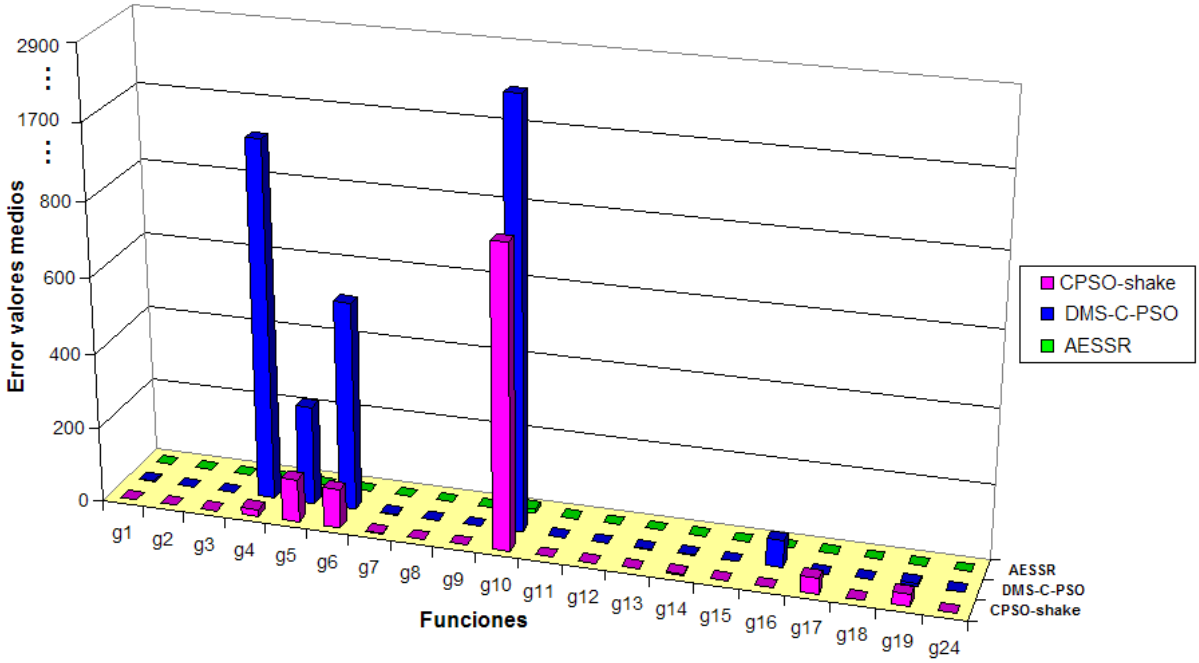


Figura 5.3: Error de valores medios con respecto a los óptimos.

tabla puede observarse que en la mayoría de las funciones existe una diferencia significativa entre los algoritmos, ya que los valores de p obtenidos son menores a 0.05. Considerando la tabla 5.3, se refleja que la diferencia favorece a CPSO-shake, el cual alcanza los óptimos o valores cercanos a éstos. No sucede lo mismo con DMS-C-PSO. Se exceptúan de la conclusión a las funciones g5, g6, g9, g11, g12 y g24, en las cuales no se detectó una diferencia estadísticamente significativa. En cada entrada de la tabla 5.6, se encuentra el valor estimado (EV) y su intervalo de confianza correspondiente del 95 %, para todas las instancias que resultaron ser estadísticamente diferentes. Estos intervalos ([LI, LS] en la tabla 5.6) se obtuvieron con el test de Tukey y describen bajo qué condiciones (rangos) las diferencias son significativas. Para las funciones g20, g21, g22 y g23 no existen valores de p ya que para estas instancias, los valores encontrados por uno o ambos algoritmos, no resultaron ser soluciones factibles, con lo cual el estudio estadístico para estas funciones carece de sentido.

En las figuras 5.5 y 5.6 pueden observarse las gráficas del test de Tukey para cada una de las instancias en las que se encontraron diferencias significativas. El algoritmo 1- hace referencia a CPSO-shake y el algoritmo 2- hace referencia a DMS-C-PSO.

Como conclusión de la comparacion a través de las tablas 5.3, 5.4 y 5.5, la robustez de CPSO-shake es un aspecto que debe ser mejorado. Con respecto a la calidad de los

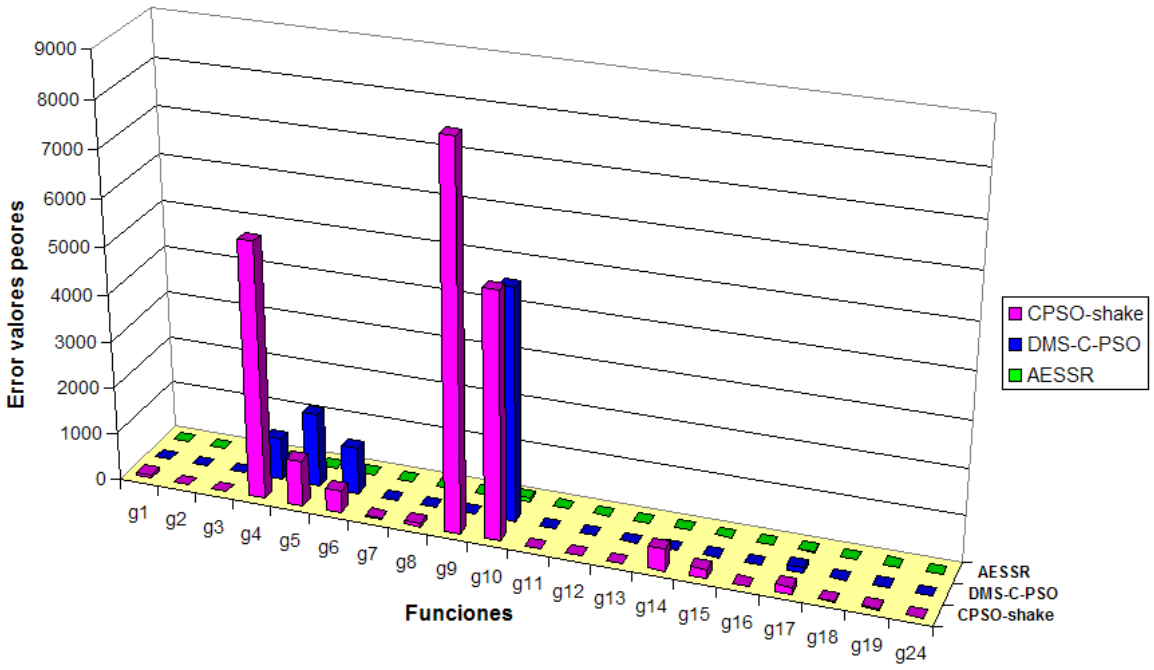


Figura 5.4: Error de valores peores con respecto a los óptimos.

resultados, CPSO-shake obtuvo un desempeño comparable con el de AESSR, el cual es un algoritmo eficaz para problemas de optimización con restricciones. CPSO-shake obtuvo mejores resultados que DMS-C-PSO y ese hecho fue confirmado en la comparación que se efectuó mediante el análisis estadístico.

Tabla 5.3: Mejores valores obtenidos con CPSO-shake y DMS-C-PSO (350,000 evaluaciones) y AESSR (500,000 evaluaciones).

Función	**Benchmark**	**CPSO-shake**	**AESSR**	**DMS-C-PSO**
g1	-15.000	**-15.000**	**-15.000**	-14.770
g2	-0.803	**-0.803**	-0.739	**-0.803**
g3	-1.000	**-1.000**	**-1.000**	-0.998
g4	-30,665.539	-30,665.538	**-30,665.539**	$-32,217,121^*$
g5	5,126.496	5,126.498	5,126.497	5,132.459
g6	-6,961.813	-6,961.825	-6,961.814	-6,961.910
g7	24.306	24.309	**24.306**	24.741
g8	-0.095	**-0.095**	-0.096	**-0.095**
g9	680.630	**680.630**	**680.630**	680.635
g10	7,049.248	7,049.285	7,049.408	7,498,872
g11	0.749	**0.749**	0.750	**0.749**
g12	-1.000	**-1.000**	**-1.000**	**-1.000**
g13	0.053	0.054	0.054	0.072
g14	-47.764	-47.635	-47.765	-47.635
g15	961.715	**961.715**	**961.715**	**961.715**
g16	-1.905	**-1.905**	**-1.905**	$-2,579^*$
g17	8,853.539	**8,853.539**	8,853.540	8,856.526
g18	-0.866	**-0.866**	**-0.866**	-0.865
g19	32.655	34.018	**32.665**	34.730
g20	$0,097^*$	$0,256^*$	-	$0,664^*$
g21	193.724	$361,846^*$	-	$253,590^*$
g22	236.430	$545,112^*$	-	$0,000^*$
g23	-400.055	-326.963	**-348.816**	$-2,294,500^*$
g24	-5.508	**-5.508**	**-5.508**	**-5.508**

* Solución infactible.

Tabla 5.4: Errores de los valores medios con respecto al benchmark.

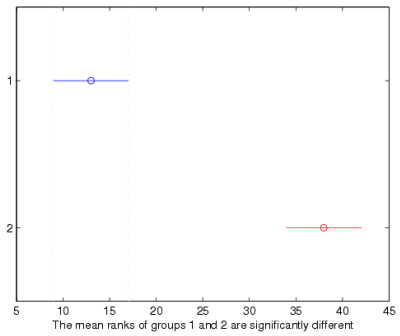
Función	**CPSO-shake**	**AESSR**	**DMS-C-PSO**
g1	0	0	1.822
g2	0.007	0.106	0.040
g3	0	0	0.011
g4	19.360	0.001	1,552.439
g5	114	0.001	263.296
g6	102.738	0.001	554.023
g7	0.606	0.001	1.576
g8	0	0.001	0.001
g9	0.743	0	0.085
g10	801.153	11.839	2,727.872
g11	0	0.001	0
g12	0	0	0
g13	0.397	0.063	0.517
g14	2.099	0.002	0.130
g15	0.801	0	2.226
g16	0.110	0	0.011
g17	41.169	0.178	70.461
g18	0.079	0.015	0.002
g19	31.850	0.174	2.074
g20	2.679	-	5.368
g21	23.673	-	104.905
g22	56.334	-	1,215.442
g23	128.212	241.879	398.948
g24	0	0	0

Tabla 5.5: Errores de los peores valores obtenidos con respecto al benchmark.

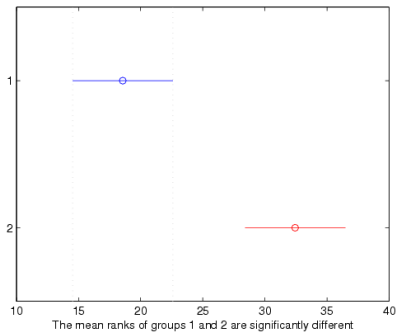
Función	**CPSO-shake**	**AESSR**	**DMS-C-PSO**
g1	73.844	0	6.710
g2	0.739	0.146	0.157
g3	0.005	0	0.037
g4	5,479.311	0.001	894.102
g5	973.142	0.001	1,575.496
g6	479.259	0.001	989.187
g7	23.095	0.006	3.577
g8	72.169	0.001	0.055
g9	8,144.502	0	0.1671
g10	5,238.894	77.71	4,951.752
g11	0.612	0.001	0.001
g12	0.282	0	0.267
g13	1.029	0.386	0.776
g14	470.155	0.011	5.764
g15	192.978	0	4.672
g16	2.084	0	1.080
g17	150.075	4.277	114.573
g18	6.304	1.541	0.704
g19	23.18	0.587	19.432
g20	6.991	-	7.095
g21	52.878	-	482.595
g22	67.920	-	7,169.869
g23	155.790	145.361	478.780
g24	0	0	0

Tabla 5.6: Test de Kruskal-Wallis para CPSO-shake y DMS-C-PSO.

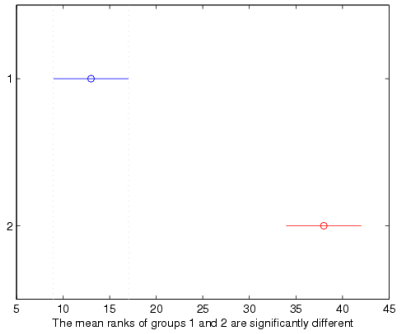
Función	p	LI	EV	LS
g1	1.32439e-009	-33.0797	-25.0000	-16.9203
g2	0.0008	-21.9591	-13.8800	-5.8009
g3	1.282e-009	-33.0653	-25.0000	-16.9347
g4	6.49995e-008	14.2001	22.2800	30.3599
g5	0.4095	-	-	-
g6	1	-	-	-
g7	9.81083e-008	-29.8160	-21.8000	-13.7840
g8	7.71695e-009	-28.1277	-21.0000	-13.8723
g9	0.8996	-	-	-
g10	0.0193	-17.7137	-9.6400	-1.5663
g11	1	-	-	-
g12	1	-	-	-
g13	0.0319	-16.9156	-8.8400	-0.7644
g14	6.29599e-005	-24.4925	-16.4400	-8.3875
g15	6.6082e-005	-24.5156	-16.4400	-8.3644
g16	3.1303e-006	10.9196	18.8400	26.7604
g17	7.8276e-005	-24.3587	-16.2800	-8.2013
g18	0.0068	-19.2411	-11.1600	-3.0789
g19	7.73009e-007	12.2855	20.3600	28.4345
g20	-	-	-	-
g21	-	-	-	-
g22	-	-	-	-
g23	-	-	-	-
g24	1	-	-	-



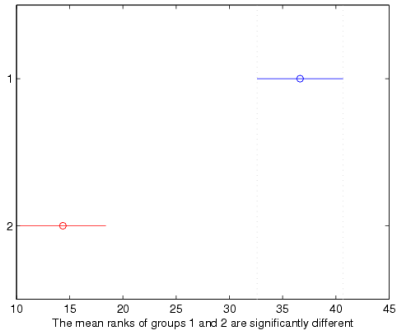
(a) g1



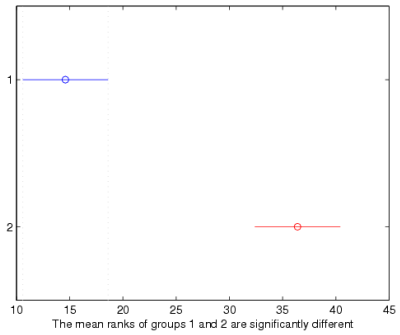
(b) g2



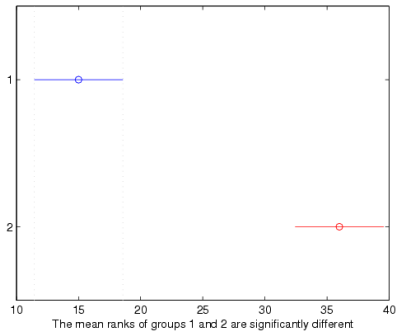
(c) g3



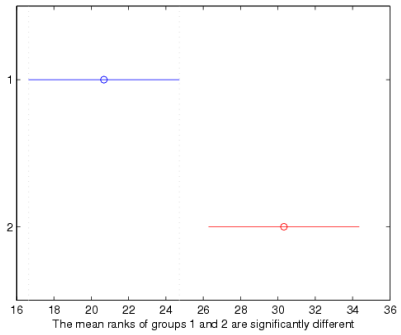
(d) g4



(e) g7

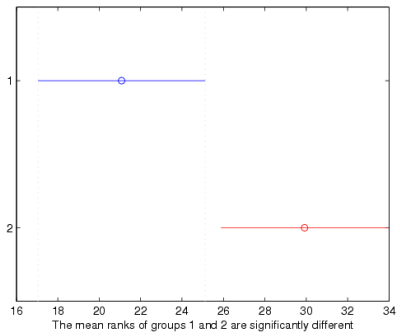


(f) g8

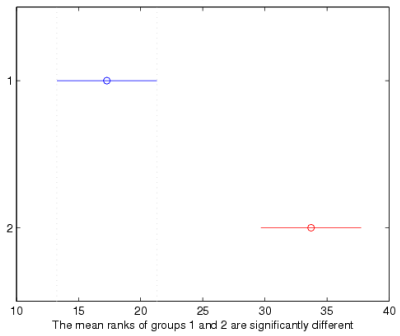


(g) g10

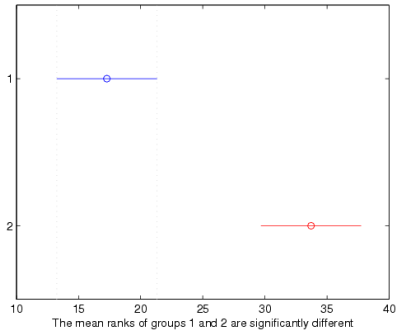
Figura 5.5: Test de Tukey para CPSO-shake (1-) y el algoritmo DMS-C-PSO (2-).



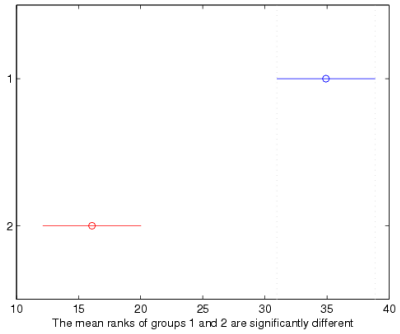
(a) g13



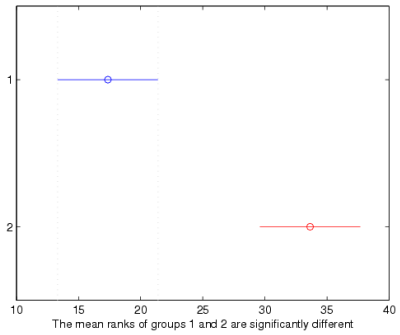
(b) g14



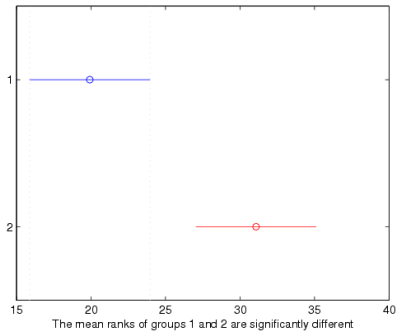
(c) g15



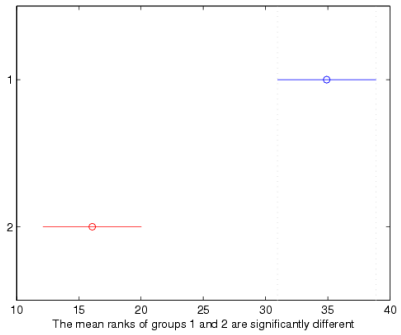
(d) g16



(e) g17



(f) g18



(g) g19

Figura 5.6: Test de Tukey para CPSO-shake (1-) y el algoritmo DMS-C-PSO (2-).

5.6. Diseño Óptimo de Armaduras

El diseño óptimo de armaduras es un tema muy estudiado en la literatura especializada [11, 95]. Una armadura es una estructura rígida, compuesta de barras conectadas por soldaduras o articulaciones libres de fricción. Las barras deben ser sometidas a cargas de tracción o compresión, con lo cual su diseño debe ser lo suficientemente fuerte como para soportar las cargas. El diseño, sin embargo, debe ser al mismo tiempo, liviano y económico.
Para este estudio se seleccionaron tres problemas de optimización de diseño de armaduras tomados de [10]: una armadura plana de 10 barras, una armadura espacial (3D) de 25 barras y una armadura plana de 200 barras. La descripción completa de estos problemas se encuentra en el Apéndice C. Nótese que los tres son problemas de minimización con varias restricciones de desigualdad.
Existe en la bibliografía especializada una buena cantidad de trabajos que optimizan estas funciones, aunque en varios casos se usan diferentes variantes de estos problemas (por ejemplo, con diferentes condiciones de carga y esfuerzo). Por tal motivo, se realizó una búsqueda exhaustiva de los trabajos cuyas descripciones de estos problemas coincidiera con las utilizadas en esta tesis. Los resultados se compararán con las siguientes técnicas:

1. Técnicas de programación matemática [10]: Direcciones Factibles (CONMIN y OPTDYN), Programación Cuadrática Recursiva de Pshenichny (LINRM), Proyección de Gradiente (GRP-UI) y Función de Penalización Exterior (SUMT).

2. Dos algoritmos genéticos simples con una función de penalización: SGA [54] y GENETICO [29].

3. Método basado en gradiente para optimización restringida embebido en un programa de computación denominado BEHSAZ (BEHP) [117].

4. Programación por Metas no Lineal (NLP) [45].

5. Harmony Search (HSA) [98].

6. Algoritmo Genético con esquema de penalización adaptativo (AP-GA) [99].

7. Versión modificada de *Pareto Archived Evolution Strategy* (PAES) [87], usada para optimización mono-objetivo con restricciones (ISPAES) [65].

8. Evolución Diferencial [147] hibridizada con un Algoritmo Cultural (CDE) [95].

9. PSO mejorado para diseño estructural (IPSO) [144].

10. Una heurística basada en PSO (HPSO) [102].

11. Algoritmo Genético hibridizado con un Sistema Inmune Artificial (AIS-GA) [11].

Se realizaron 30 ejecuciones independientes por problema con un total de 100,100 evaluaciones de la función objetivo por ejecución para las armaduras de 10 y de 200 barras. Este valor fue seleccionado debido a que es el más bajo reportado (ver por ejemplo [95]). Exceptuando CDE, las demás técnicas emplean un mayor número de evaluaciones. IPSO utiliza 8,000 evaluaciones en 20 ejecuciones independientes para la armadura de 25 barras. CPSO-shake fue probado con dichas cantidades (8,000 en 20 ejecuciones) pero también con 100,100 en 30 ejecuciones, para verificar la robustez del algoritmo. La cantidad de evaluaciones empleadas por SUMT, así como el conjunto de parámetros con los que se obtuvieron los resultados reportados en [10], no se encuentran disponibles. Por ello se emplearon 100,100 evaluaciones. El detalle de los parámetros utilizados por CPSO-shake y por IPSO se resumen en la tabla 5.7. La justificación de la elección de los valores de los parámetros de CPSO-shake se resume en el Capítulo 7.

Tabla 5.7: Configuración de CPSO-shake e IPSO para funciones de ingeniería estructural.

Alg.	**Pobl.**	**ProbMut**	$\mathcal{C}_i$	$\mathcal{X}$	**Vecindario**	**ProbActGauss**	w
CPSO-shake	10	0.00	1.70	0.70	3	0.074	
IPSO	40	-	[0,4]*	-	-	-	[0.35,1]*

* Variable en dicho rango.

En las tablas 5.8, 5.9 y 5.10 se muestran los valores de las variables de decisión, correspondientes a las mejores soluciones obtenidas para cada problema, con CPSO-shake. En la tabla 5.11 se presenta la comparación de los resultados (valores de las función objetivo) obtenidos con CPSO-shake y las técnicas reportadas previamente. Nótese que no todas las aproximaciones reportan resultados para los tres problemas. Se hace notar también que los valores de CDE reportados en [95] para la armadura de 200 barras no se incluyen porque el conjunto de datos de entrada difiere del utilizado por los demás algoritmos. De los valores mostrados en la tabla 5.11 se concluye que, CPSO-shake es el que obtiene mejor valor objetivo para cada uno de los problemas.

En la tabla 5.12 se resume el mejor valor conocido reportado por alguna de las técnicas, así como la desviación estándar (cuando se reporta) y el peor valor alcanzado para cada uno de los problemas. Nótese que todas las soluciones encontradas son factibles. Para el caso de la armadura de 10 barras, el mejor resultado conocido es el obtenido por el algoritmo cultural (CDE) [95]. Para la armadura de 25 barras, el mejor resultado conocido es el obtenido con un algoritmo basado en PSO (IPSO) [144]. Finalmente, para el caso de la armadura de 200 barras el mejor resultado conocido es el obtenido por una técnica de programación matemática con función de penalización (SUMT) [10]. CPSO-shake obtiene valores mejores que los óptimos conocidos para los tres problemas. Para las armaduras de 25 y 200 barras las desviaciones estándar no se comparan porque no se dispone de esos datos para las demás técnicas. Para el problema de la armadura de 10 barras, CDE obtiene el mejor valor de desviación estándar y alcanza un mejor valor para la peor solución, puesto que es menor que el obtenido por CPSO-shake. Para la armadura de 25 barras CPSO-shake obtiene el peor inferior que el alcanzado por IPSO.

Tabla 5.8: Vector solución para la armadura plana de 10 barras.

	Valor
x_0	104.714546
x_1	88.240891
x_2	0.400003
x_3	0.400000
x_4	120.564369
x_5	46.732246
x_6	98.870956
x_7	19.894732
x_8	0.400006
x_9	0.400001
x_{10}	3.411510
x_{11}	0.400211
x_{12}	63.583042
x_{13}	4.455610
x_{14}	42.467812
x_{15}	74.717102
x_{16}	136.491806
x_{17}	30.628040
x_{18}	0.400001
x_{19}	0.400000
Desplazamiento	7.591675
Esfuerzo	84.707460
Peso	4,656.361619

Tabla 5.9: Vector solución para la armadura espacial de 25 barras con 8,000 evaluaciones.

	Valor
x_0	0.103047
x_1	0.100770
x_2	3.521567
x_3	0.101640
x_4	1.897667
x_5	0.772679
x_6	0.164298
x_7	3.994101
Desplazamiento	1.562338
Esfuerzo	92,119.015262
Peso	467.581298

Se concluye, a través de la comparación indirecta de resultados, que para la armadura de 10 barras la mejora es casi imperceptible, para la armadura de 25 barras existe una

Tabla 5.10: Vector solución para la armadura plana de 200 barras.

	Valor		Valor
x_0	0.099778	x_{16}	0.059441
x_1	0.827075	x_{17}	50.606869
x_2	0.010058	x_{18}	0.010290
x_3	0.010004	x_{19}	66.012398
x_4	3.645809	x_{20}	0.279517
x_5	0.049748	x_{21}	0.746613
x_6	0.010007	x_{22}	94.119659
x_7	10.451566	x_{23}	0.488295
x_8	0.010109	x_{24}	114.492180
x_9	16.537201	x_{25}	1.489903
x_{10}	0.090882	x_{26}	24.173244
x_{11}	0.057575	x_{27}	69.310654
x_{12}	23.390146	x_{28}	168.946198
x_{13}	0.175855	Desplazamiento	45.352070
x_{14}	34.062832	Esfuerzo	1,105.103661
x_{15}	0.236381	Peso	22,705.327292

Tabla 5.11: Mejores valores obtenidos por CPSO-shake y otras técnicas.

Algoritmo	**Arm. de 10 barras**	**Arm. de 25 barras**	**Arm. de 200 barras**
CONMIN [10]	4,793.00	-	34,800.00
OPTDYN [10]	9,436.00	-	-
LINRM [10]	6,151.00	-	33,315.00
GRP-UI [10]	5,077.00	-	-
SUMT [10]	5,070.00	-	27,564.00
SGA [54]	4,987.00	-	-
BEHP [117]	4,981.10	-	-
GENETICO [29]	5,691.82	539.48	-
NLP [45]	5,013.24	-	-
HSA [98]	5,057.88	544.38	-
AP-GA [99]	5,069.09	-	-
ISPAES [65]	5,951.00	569.80	-
CDE [95]	4,656.39	-	-
IPSO [144]	5,024.21	485.33	-
HPSO [102]	5,060.92	545.19	-
AIS-GA [11]	5,062.67	-	-
CPSO-shake	**4,656.36**	**467.58**	**22,705.32**

mejora del 4 % aproximadamente, y la mayor diferencia se observa en la armadura de 200 barras, en donde la mejora al óptimo conocido asciende a 18 %.

Una comparación directa no es factible debido a que no se consiguieron los archivos fuente de alguno de los métodos que resuelven estos problemas, pero observando los valores obtenidos con CPSO-shake, no existe duda de que el desempeño del algoritmo es bueno con respecto a la mejor solución hallada previamente. En cuanto a la robustez del algoritmo, es conveniente realizar un estudio estadístico de la distribución de los datos en la muestra de las soluciones encontradas.

Tabla 5.12: Comparación de resultados de CPSO-shake con los mejores valores reportados.

	Mejores Conocidos			**CPSO-shake**		
Armadura	**Mejor**	**Desv. Est.**	**Peor**	**Mejor**	**Desv. Est.**	**Peor**
10 barras	4,656.39	0.18	4,656.71	**4,656.36**	2.84	4,696.06
25 barras	485.33	-	534.84	**467.30**$^{+}$	0.35	470.87
				467.58*	5.36	480.27
200 barras	27,564.00	-	-	**22,705.32**	1,566.65	30,107.62

$^{+}$ 100,100 ev.

* 8,000 ev.

Se realizaron Cajas de Tukey (*Box Plots*) [22] y Diagramas de Dispersión (*Scatter Plots*) [22] para cada una de las funciones evaluadas. Las Cajas de Tukey son gráficos representativos de las distribuciones, a través de las cuales se pueden detectar las características de los diferentes grupos involucrados en la muestra [22]. Generalmente estos gráficos resumen en 5 valores las propiedades de la muestra (o población) de datos. Como se observa en la figura 5.7, la *localización* de los datos está representada con la mediana dibujada dentro de la caja. La *dispersión* de los datos se puede observar en la altura de la caja más la distancia de los bigotes (representados con líneas discontinuas externas a la caja) si existieran. En la caja se ubica el primer cuartil (el 25 % de los datos), la mediana y, luego, el tercer cuartil (el 75 % de los datos). El *sesgo* de los datos se puede observar en la desviación que existe entre la mediana y el centro de la caja, como así también en la relación existente entre las longitudes de los bigotes. Las *colas* de la distribución se detectan por la longitud de los bigotes con relación a la altura que posea la caja. Finalmente, los valores que se alejan mucho de la mayoría de valores en la muestra, se marcan fuera de los extremos de los bigotes y se denominan *outliers*.

Los Diagramas de Dispersión permiten visualizar de manera simple cómo están distribuidos los datos en la muestra. Obsérvese que, en las figuras que se muestran a continuación, los diagramas de dispersión se encuentran anexos a la izquierda (en posición vertical) a los gráficos de las cajas de Tukey. La secuencia de números en el eje de las y indica los valores correspondientes al rango de la muestra, los cuales coinciden tanto para la caja de Tukey como para el diagrama de dispersión.

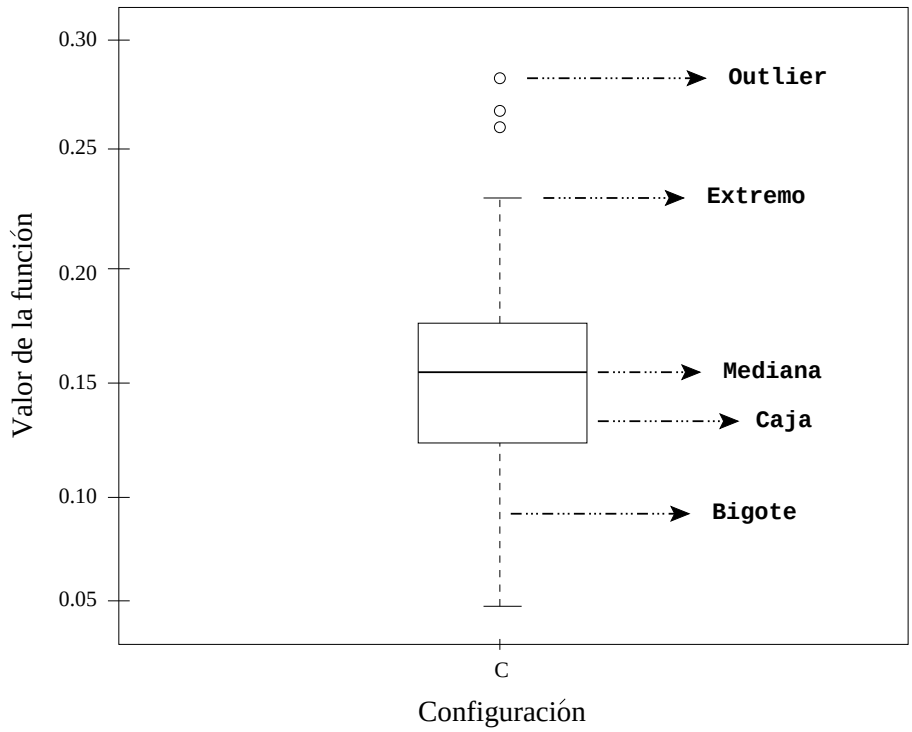


Figura 5.7: Herramienta gráfica Cajas de Tukey.

La figura 5.8 muestra la caja de Tukey y el diagrama de dispersión para la armadura de 10 barras. La caja indica que la localización de los datos en la muestra es equilibrada ya que la mediana se ubica en el centro de la caja. Existe dispersión de los datos aunque, juzgando el rango del primer y tercer cuartil, ésta no es exagerada. Se pueden observar colas en la distribución pero no outliers. El diagrama de dispersión evidencia la diversidad de soluciones que obtuvo el algoritmo y, aunque el rango no es extenso, este hecho interfiere con la robustez del algoritmo.

En la figura 5.9 se puede observar que la caja de Tukey para la armadura de 25 barras posee un sesgo hacia la derecha en la distribución de datos y la dispersión de datos se encuentra acotada en un rango pequeño. Algunos valores outliers se detectan fuera de la caja. Obsérvese que, no existen bigotes producto de la ausencia de colas en la distribución. El diagrama de dispersión evidencia que la mayoría de valores en la muestra se encuentran agrupados, concluyéndose que el algoritmo fue más robusto para este problema. Nótese que, para este estudio estadístico se utilizó una muestra de 30 ejecuciones, para un adecuado estudio de la robustez del algoritmo.

La figura 5.10 muestra información sobre la distribución de los mejores valores obtenidos, para la armadura plana de 200 barras. La caja de Tukey evidencia un sesgo en los datos con una dispersión considerable, posiblemente debida a la dificultad del algoritmo para encontrar buenos valores. La presencia de bigotes muestra la existencia de colas en la distribución de los datos y dos valores outliers se alejan de la población. En el diagrama

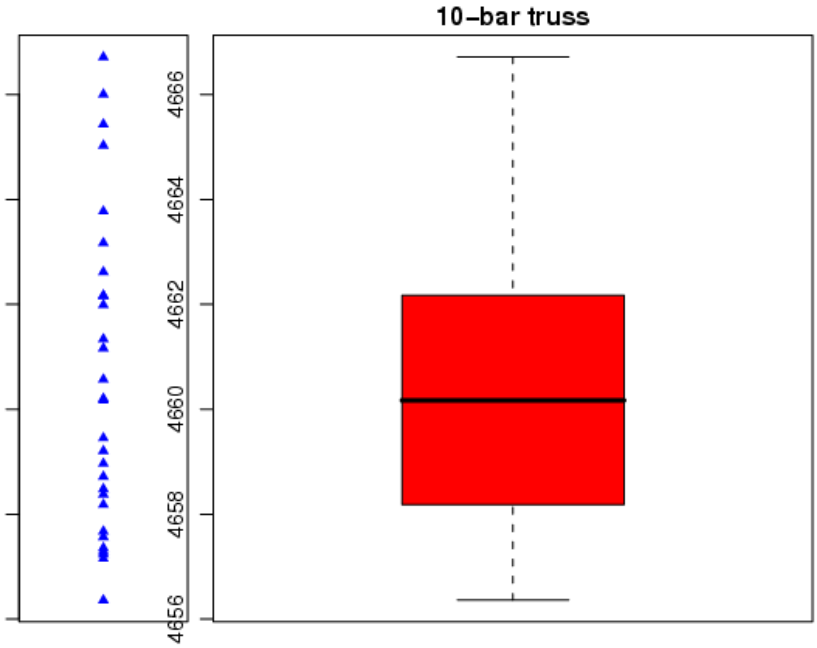


Figura 5.8: Caja de Tukey y diagrama de dispersión para la armadura plana de 10 barras.

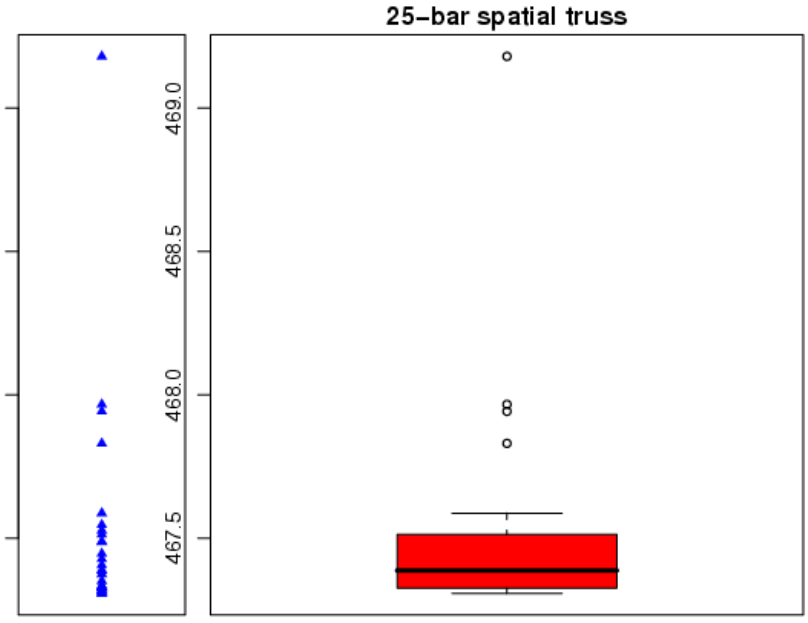


Figura 5.9: Caja de Tukey y diagrama de dispersión para la armadura espacial de 25 barras.

de dispersión se observa cómo los datos se concentran en la mitad del rango de valores posibles mostrando que el algoritmo encuentra, en general, los óptimos con dichos valores.

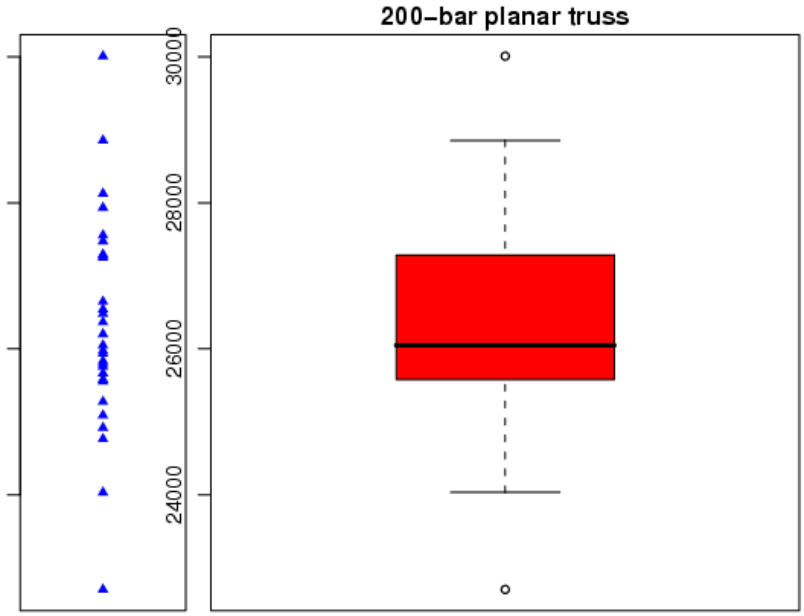


Figura 5.10: Caja de Tukey y diagrama de dispersión para la armadura plana de 200 barras.

Acerca del desempeño de CPSO-shake para problemas de diseño óptimo de armaduras, puede concluirse que el algoritmo encontró soluciones que superan a los óptimos conocidos. No obstante, las causas que degradan su robustez, deberán ser consideradas en estudios posteriores. Comparado con el caso de prueba de las 24 funciones del benchmark, CPSO-shake mejoró su desempeño. Esto se debe a la alta capacidad de exploración del espacio de búsqueda, a pesar de que esto mismo puede producir algunas soluciones malas. Esto interfiere directamente en la robustez de CPSO-shake. En otras palabras, la ventaja de mayor exploración propia de CPSO-shake, produce que la obtención de buenas soluciones deba equilibrarse con el deterioro de la robustez del algoritmo.

5.7. Diseño Ingenieril de Piezas

Al igual que en el caso de los problemas de diseño óptimo de armaduras, los problemas de diseño ingenieril son normalmente utilizados para mostrar la efectividad de los algoritmos de optimización con restricciones. En general, poseen variables de diseño con características mezcladas (continuas y discretas), funciones objetivo y restricciones no lineales que pueden estar activas en el óptimo. Diferentes métodos han sido empleados para resolver este tipo de problemas tales como Branch and Bound con Programación Cuadrática Secuencial [163], Programación Cuadrática Recursiva [20], Algoritmo de Linealización Secuencial [109], Programación no lineal entera-discreta-continua [79], Programación Mixta Discreta no Lineal [101], Recocido Simulado [203], Algoritmos Genéticos [198], Programación Evolutiva [18] y Estrategias Evolutivas [178].
Para evaluar el desempeño de CPSO-shake, se seleccionaron 4 problemas de optimización de diseño ingenieril tomados de la literatura especializada, E01: diseño óptimo de una viga soldada [148] (*Welded Beam*), E02: diseño óptimo de una válvula de presión [163] (*Pressure Vessel*), E03: diseño óptimo de un reductor de velocidad [62] (*Speed Reducer*) y E04: diseño óptimo para la tensión/compresión de un resorte [5] [10] (*Tension/Compression Spring*). La definición de cada problema se encuentra en el Apéndice D.
En todos los experimentos se efectuaron 30 ejecuciones independientes para resolver cada problema, utilizando 24,000 y 30,000 evaluaciones por ejecución. La justificación de esta decisión fue la de evaluar el algoritmo con un número considerablemente menor al empleado por otros algoritmos (usualmente 27,000 y 30,000) para determinar si CPSO-shake podía alcanzar el óptimo en menos evaluaciones. También fue necesario realizar pruebas con 30,000 evaluaciones para verificar la robustez del algoritmo ya que con una cantidad menor de evaluaciones la robustez generalmente decae. Los valores de los parámetros con los que se ejecutó CPSO-shake son los mismos que los empleados en el apartado anterior (diseño óptimo de armaduras) y se indican en la tabla 5.13.

Tabla 5.13: Configuración de PSO para funciones de ingeniería.

Población	**ProbMutMin**	**ProbMutMax**	$\mathcal{C}_i$	$\mathcal{X}$	**Vecindario**	**ProbActGauss**
10	0.00	0.00	1.70	0.70	3	0.074

Los resultados de CPSO-shake fueron comparados con los mejores reportados en la literatura hasta el momento de la realización de este trabajo de tesis. Esos valores se obtuvieron con dos algoritmos, una versión del algoritmo PSO (referenciado como COPSO) y una variante de Evolución Diferencial (referenciado como DE). COPSO [66] es un algoritmo PSO que utiliza dos operadores de perturbación para evitar la convergencia prematura, y que a través de un archivo externo, actualiza los factores de tolerancia a fin de extender la supervivencia de los individuos conservados en el archivo. Los autores obtuvieron los mejores resultados luego de realizar 30,000 evaluaciones de cada función objetivo. DE [121] es un algoritmo basado en Evolución Diferencial que identifica y mantiene soluciones infactibles cerca de la zona factible para que, a través de la utilización

de operadores genéticos, pueda crear nuevos individuos dentro de la zona factible. Los resultados con DE fueron obtenidos después de realizar 30,000 evaluaciones de la función objetivo.
Las parámetros relevantes empleados por COPSO y DE se encuentran detallados en la tabla 5.14.

Tabla 5.14: Configuración de COPSO y DE.

Alg.	**Población**	w	$\mathcal{C}_i$	**Prob. Crossover**	**Prob. Mutación**
COPSO	100	1.0	[0.5,1]*	-	-
DE	100	-	-	0.6	0.01

* aleatorio en el rango.

La tabla 5.15 muestra los mejores resultados conocidos, y los mejores encontrados por cada algoritmo, luego de 24,000 evaluaciones de las funciones objetivo. Es importante remarcar que dichos valores se obtienen también utilizando 30,000 evaluaciones.
Para E01 los tres algoritmos obtuvieron el mejor valor conocido. Para E02 y E03 sólo CPSO-shake y COPSO alcanzaron este valor, mientras que DE se aproximó bastante a dichos valores. Para el problema E03 no se reporta el mejor resultado conocido y considerando los valores obtenidos con cada algoritmo, CPSO-shake obtuvo el valor más bajo (el cual es el mejor ya que se trata de un problema de minimización). COPSO alcanzó un valor levemente peor que CPSO-shake y DE obtuvo una solución infactible.

Tabla 5.15: Mejores valores obtenidos con cada algoritmo.

Problema	**Benchmark**	**CPSO-shake**	**COPSO**	**DE**
E01	1.724852	**1.724852**	**1.724852**	**1.724852**
E02	6,059.714335	**6,059.714335**	**6,059.714335**	6,059.7143
E03	N/A	**2,996.348165**	2,996.372448	2,996.348094*
E04	0.012665	**0.012665**	**0.012665**	0.012689

* Solución infactible. N/A: Información no disponible.

La tabla 5.16 muestra los valores medios alcanzados por los algoritmos sobre el total de 30 ejecuciones independientes con 30,000 evaluaciones. Como puede observarse, en general, los mejores valores medios fueron obtenidos por COPSO excepto para la función E03 cuya media es levemente inferior para CPSO-shake. Nótese que para que esta comparación de valores sea justa, se utilizan las medias sobre las 30 ejecuciones con 30,000 evaluaciones así como lo reporta COPSO y DE.

La tabla 5.17 muestra las desviaciones estándar alcanzadas por los algoritmos sobre el total de 30 ejecuciones independientes con 30,000 evaluaciones de la función objetivo por cada ejecución. Obsérvese que los valores más bajos para E01 y E04 fueron obtenidos

Tabla 5.16: Valores medios obtenidos por los algoritmos.

Problema	**CPSO-shake**	**COPSO**	**DE**
E01	1.8540	1.7248	1.7776
E02	6,089.5091	6,071.0133	6,379.9380
E03	2,996.3482	2,996.4085	2,996.3480*
E04	0.0130	0.0126	0.0131

* Solución infactible.

por COPSO, pero para E02 y E03 CPSO-shake obtuvo mejores valores de desviaciones estándar.

Tabla 5.17: Desviaciones estándar obtenidas por los algoritmos.

Problema	**CPSO-shake**	**COPSO**	**DE**
E01	0.0194	1.2E-05	8.8E-02
E02	10.0256	15.1011	210.0000
E03	0.0000	0.0286	0.0000*
E04	3.1E-04	1.2E-06	3.9E-04

* Solución infactible.

En las tablas 5.18, 5.19, 5.20 y 5.21 se pueden observar los valores de los vectores solución obtenidos con CPSO-shake junto a sus correspondientes vectores de restricciones, para cada uno de los cuatro problemas de ingeniería.

Tabla 5.18: Vector solución para E01 (*welded beam*) obtenido por CPSO-shake.

Variable	Valor
x_1	0.205729
x_2	3.470488
x_3	9.036624
x_4	0.205729
$g_1(\vec{x})$	-1.819E-12
$g_2(\vec{x})$	-0.003721
$g_3(\vec{x})$	0.000000
$g_4(\vec{x})$	-3.432983
$g_5(\vec{x})$	-0.080729
$g_6(\vec{x})$	-0.235540
$g_7(\vec{x})$	0.000000
$f(\vec{x})$	1.724852

Como conclusión de la comparación efectuada, se puede decir que CPSO-shake, pese a ser un algoritmo PSO más simple que COPSO, encontró los óptimos en la totalidad

Tabla 5.19: Vector solución para E02 (*pressure vessel*) obtenido por CPSO-shake.

Variable	Valor
x_1	0.812500
x_2	0.437500
x_3	42.098445
x_4	176.636595
$g_1(\vec{x})$	-4.500E-15
$g_2(\vec{x})$	-0.035880
$g_3(\vec{x})$	-1.164E-10
$g_4(\vec{x})$	-63.363404
$f(\vec{x})$	6,059.714335

Tabla 5.20: Vector solución para E03 (*speed reducer*) obtenido por CPSO-shake.

Variable	Valor
x_1	3.500000
x_2	0.700000
x_3	17
$x4$	7.300000
x_5	7.800000
x_6	3.350214
x_7	5.286683
$g_1(\vec{x})$	-0.073915
$g_2(\vec{x})$	-0.197998
$g_3(\vec{x})$	-0.499172
$g_4(\vec{x})$	-0.901471
$g_5(\vec{x})$	0.000000
$g_6(\vec{x})$	-5.000E-16
$g_7(\vec{x})$	-0.702500
$g_8(\vec{x})$	-1.000E-16
$g_9(\vec{x})$	-0.583333
$g_{10}(\vec{x})$	-0.051325
$g_{11}(\vec{x})$	-0.010852
$f(\vec{x})$	2,996.348165

de problemas aunque con una desviación estándar un poco mayor, en al menos dos de los problemas (con respecto a las de COPSO). Pero es importante remarcar que CPSO-shake obtuvo también los mejores conocidos, con un número considerablemente menor de evaluaciones. En general, los valores medios que obtuvo CPSO-shake son levemente superiores que los de COPSO, con lo cual se puede decir que la robustez de CPSO-shake es sólo levemente inferior a la de COPSO. Con respecto a DE, CPSO-shake lo supera en

Tabla 5.21: Vector solución para E04 (*tension/compression spring*) obtenido por CPSO-shake.

Variable	Valor
x_1	0.051583
x_2	0.354190
x_3	11.438675
$g_1(\vec{x})$	-2.000E-16
$g_2(\vec{x})$	-1.000E-16
$g_3(\vec{x})$	-4.048765
$g_4(\vec{x})$	-0.729483
$f(\vec{x})$	0.012665

desempeño en los cuatro problemas.
Una comparación directa entre los algoritmos no es posible debido a que no se consiguieron los archivos fuente de los métodos que resuelven estos problemas, pero observando los valores obtenidos con CPSO-shake, no existe duda que el desempeño del algoritmo es muy bueno. En cuanto a la robustez del algoritmo, resulta conveniente realizar un estudio estadístico de la distribución de los datos en la muestra de soluciones, a fin de analizar en más detalle las características de los mejores valores obtenidos en las 30 ejecuciones realizadas.

La figura 5.11 muestra la caja de Tukey y el diagrama de dispersión para el problema E01. Como se puede observar la localización de los datos es desproporcionada, visible en la mediana que se encuentra en el tercer cuartil. Se observa también una amplia dispersión de datos y colas en la distribución. Los valores distribuidos en el rango mostrado en el diagrama de dispersión presupone una robustez débil del algoritmo para este problema.
Tanto para E02 (figura 5.12) como para E03 (figura 5.13) se observa la robustez ideal del algoritmo en una caja de Tukey sin altura en donde la mediana es lo único visible. Para el caso de E02 se observa un valor outlier, no siendo así para E03. Los diagramas de dispersión muestran efectivamente que los valores de las 30 ejecuciones se centralizan en un único valor del rango (E03), y en dos valores del rango para el caso de E02.
Para E04 la figura 5.14 muestra una caja de Tukey de altura casi imperceptible evidenciando el buen comportamiento del algoritmo en la búsqueda de soluciones para este problema. Se obsevan 2 valores outliers, los cuales como se puede apreciar en el diagrama de dispersión, se encuentran alejados de la mayor parte de las soluciones encontradas.

Como conclusión, CPSO-shake resultó tener un muy buen desempeño en cuatro problemas de diseño ingenieril de piezas. Tanto a través de la comparación de valores mejores, medios y desviaciones, como a través del estudio estadístico de la muestra de soluciones, CPSO-shake resultó ser robusto para este tipo de problemas tal y como lo muestran los resultados en las tablas y las figuras 5.12, 5.13 y 5.14. Para E01 la robustez no es visible en la gráfica de la caja de Tukey, pero los resultados indican que CPSO-shake, a pesar de ser un algoritmo mucho más sencillo que COPSO y DE, logró encontrar el óptimo con un menor número de evaluaciones. Nótese que CPSO-shake empleó sólo 24,000 evaluaciones

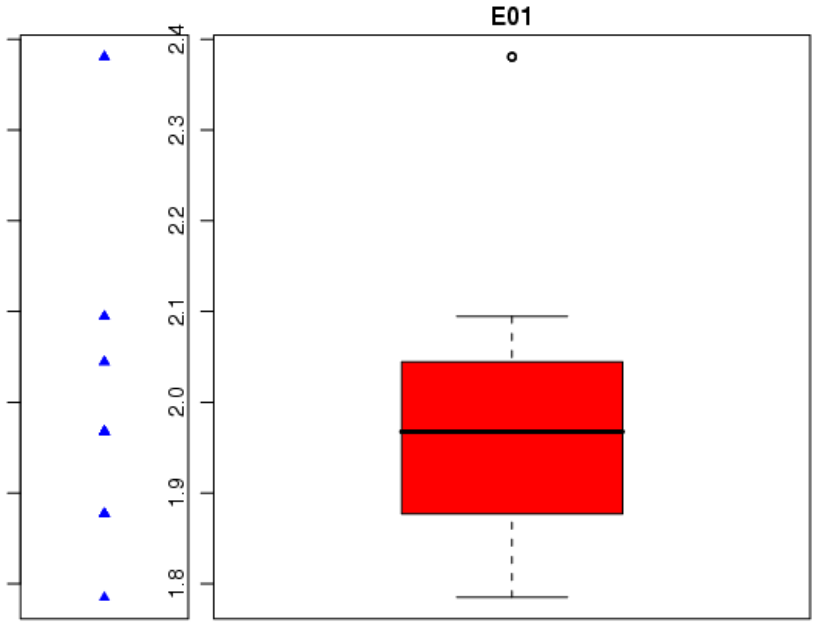


Figura 5.11: Caja de Tukey y diagrama de dispersión para E01.

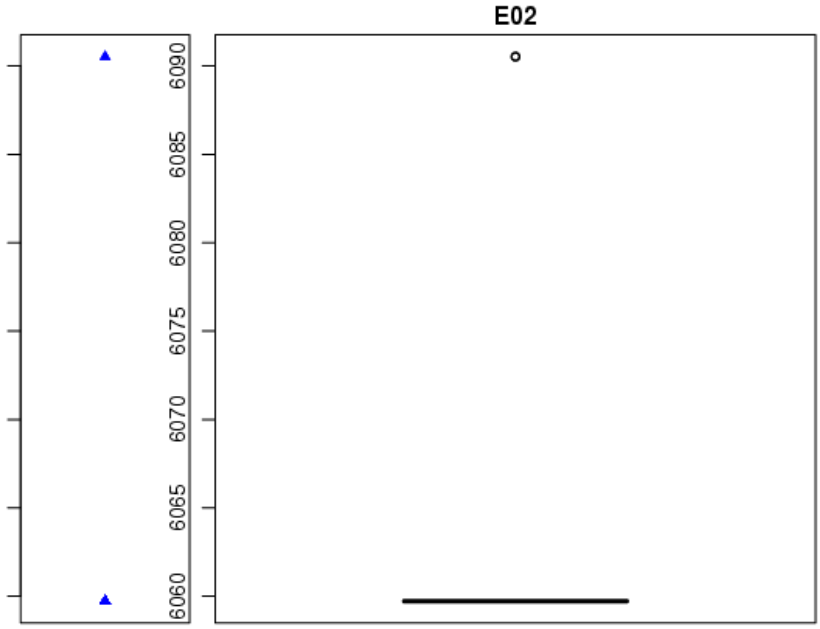


Figura 5.12: Caja de Tukey y diagrama de dispersión para E02.

de la función objetivo mientras que COPSO y DE requirieron 30,000.

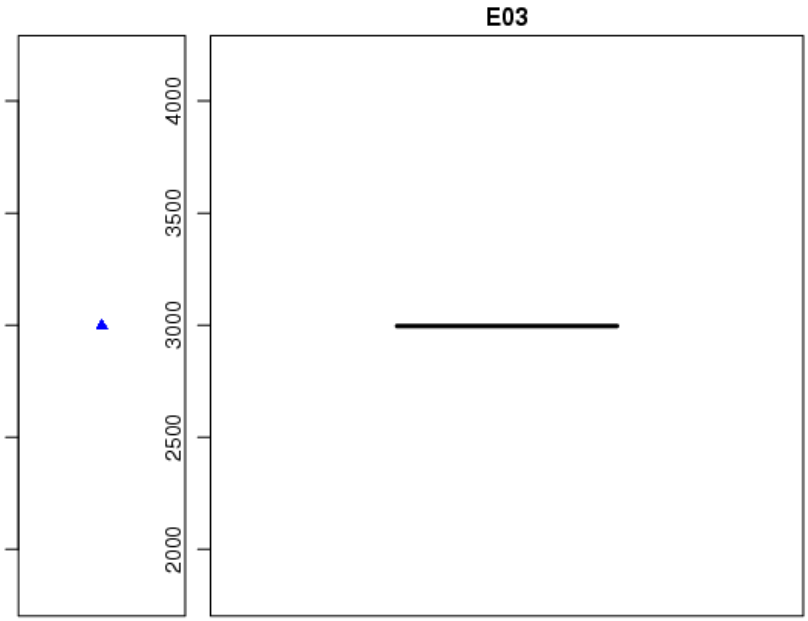


Figura 5.13: Caja de Tukey y diagrama de dispersión para E03.

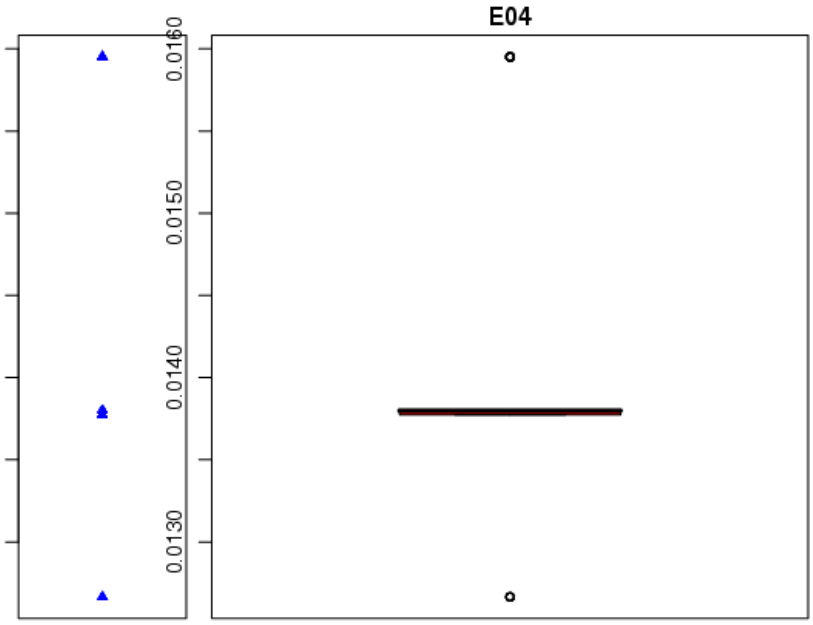


Figura 5.14: Caja de Tukey y diagrama de dispersión para E04.

5.8. Un problema de Despacho de Cargas Eléctricas

El problema de despacho de cargas eléctricas (ED por las siglas en inglés de *Economic Dispatch*) es comúnmente utilizado en la vida real y constituye una tarea importante que debe ser tomada en cuenta en la operación de los sistemas de potencia. La planificación de tareas (*schedule*) que resuelve un problema de ED representa las salidas de los generadores para cada unidad de potencia funcionando. Si se considera un horizonte de tiempo, el problema de ED es nombrado como dinámico. Si el tiempo no es una variable del problema entonces el problema de ED es considerado estático. En este trabajo sólo se considera el caso estático, ya que el caso dinámico es una extensión del primero.
El objetivo fundamental en el problema de despacho de cargas eléctricas es la planificación óptima de las unidades de generación de electricidad, de forma tal que se entregue el total de potencia requerido y se minimice el costo total de combustible (la suma de los costos de cada unidad) mientras algunas restricciones son satisfechas. Además, la generación de la planificación debe ser tan rápida como sea posible ya que es un problema real que debe ser resuelto en tiempo acotado. La descripción completa del problema se encuentra en el Apéndice E.
Una gran variedad de métodos han sido propuestos para la resolución del problema de ED tales como: técnicas de programación matemática [106], el método de iteración lambda [8], programación cuadrática [49] y métodos de gradiente [129]. Algunas aproximaciones tradicionales como Programación Dinámica [197], relajación Lagrangiana [68] y Programación Lineal [174], fueron usadas también. Algoritmos de búsqueda estocásticos como Redes Neuronales [139], Algoritmos Genéticos [100], Programación Evolutiva [140], Recocido Simulado [134], Búsqueda Tabú [107] y PSO [192] se han usado también, para resolver este problema.
Aunque existen algunos métodos que resuelven el problema de ED, éste fue seleccionado con el objetivo de verificar si CPSO-shake puede resolver un problema real con eficiencia, tanto en la calidad de la solución, como en el tiempo que emplea en encontrarla.

Se utilizaron tres instancias del problema de ED: el **Caso A** es un problema tradicional con un sistema de tres unidades y una función suave (*smooth*) [197], el **Caso B** es un sistema no suave (*non-smooth*) estándar de tres unidades que considera una función *valve point loading* [194], y el **Caso C** consiste de un sistema de 40 generadores con una función no suave (*non-smooth*) [172]. La descripción de los datos de cada caso se encuentra en el Apéndice E.

Los experimentos para validar CPSO-shake fueron ejecutados en una computadora personal con un procesador AMD Athlon de 1.24GHz con 512MB de memoria RAM. Este dato es relevante porque el costo computacional requerido para obtener los resultados es casi tan importante como el costo de la planificación. Nótese que el tiempo de respuesta es crucial en este tipo de problemas, particularmente cuando el problema es tan complejo como lo es el **Caso C**.
Los valores de los parámetros utilizados por CPSO-shake son los mismos que los de la sección anterior y se repiten en la tabla 5.22. La justificación de la selección de los mismos

se encuentra en el Capítulo 7. Los parámetros de las técnicas empleadas para la comparación indirecta del desempeño de CPSO-shake, son mostrados en la tabla 5.23. No se cuenta con la información de parámetros utilizados por el algoritmo basado en el método numérico (NM) [197] y la programación evolutiva mejorada (IEP) [140].

Tabla 5.22: Configuración de CPSO-shake para el problema de despacho de energía.

Población	**ProbMutMin**	**ProbMutMax**	$\mathcal{C}_i$	$\mathcal{X}$	**Vecindario**	**ProbActGauss**
10	0.00	0.00	1.70	0.70	3	0.074

Tabla 5.23: Parámetros de las técnicas empleadas en la comparación del desempeño de CPSO-shake.

Alg.	**Factor**	**Población**	$\mathcal{C}_i$	w	**Paso**	**Pr.Crossov.**	**Pr.Mut.**
MHNN [139]	Peso A=0.4,B=0.05	-	-	-	-	-	-
MPSO [138]	-	20	2	[0.5,1]*	0.31	-	-
IPSO [176]	-	$10/20/60^{+}$	2	[0.4,0.9]*	-	-	-
GA [194]	-	100	-	-	-	0.95	0.01
EP [199]	Escala β =0.1	50	-	-	-	-	-
TM [108]	Constricción η =0.9	-	-	-	0.01	-	-
CEP [172]#	Escala β =0.05	60	-	-	-	-	-

* Variable en dicho rango.
\+ para Caso A, B y C respectivam.
\# Idem para FEP, MFEP,IFEP [172].

- **Caso A**.
  En este caso se utilizaron 3,000 evaluaciones de la función objetivo en 4 ejecuciones independientes, según lo empleado por el algoritmo IPSO, reportado en [176]. Se utilizó dicha cantidad para poder realizar una comparación justa entre CPSO-shake y el algoritmo propuesto en [176]. Como 4 ejecuciones es una cantidad demasiado limitada, se realizaron también 10 ejecuciones (en un estudio paralelo) para poder efectuar un estudio estadístico más fiable. Tanto para las 4 como para las 10 ejecuciones, CPSO-shake obtuvo el óptimo conocido con desviación estándar de cero. La potencia demandada para este caso es de 850 MW y la mejor solución conocida [197] es \$8,194.35612. En la tabla 5.24 se presenta una comparación de resultados con respecto a los obtenidos por diferentes técnicas: método numérico [197] (NM), Red Neuronal Hopfield modificada [139] (MHNN), programación evolutiva mejorada [140] (IEP), Particle Swarm Optimization modificado [138] (MPSO) y Particle

Swarm Optimization integrado [176] (IPSO). En la tabla 5.24 se puede observar el método empleado en la columna 1; la columna 2 muestra los valores de potencia generados por cada unidad; en la columna 3 se observa el tiempo promedio, en segundos, insumidos por una ejecución; la columna 4 indica la potencia generada por el sistema. Finalmente, se muestra el costo total en $, obtenido como solución de cada método. Nótese que N/A indica la no disponibilidad de los datos para este caso. Todos los métodos comparados, excepto por MHNN (presenta una solución no factible porque no cumple la demanda de potencia), alcanzaron el total de potencia demandada con el mínimo costo conocido. El tiempo computacional promedio requerido para obtener el costo óptimo es un factor crítico a la hora de comparar soluciones de problemas del mundo real. Observando la tabla 5.24, sólo IPSO y CPSO-shake pueden ser comparados de manera justa ya que ambos fueron ejecutados en computadoras con similares características. Para este caso, CPSO-shake resultó ser mucho más veloz que IPSO.

- **Caso B**.
  Para este caso se utilizaron 6,000 evaluaciones de la función objetivo en 4 ejecuciones independientes, para poder comparar de manera justa con el algoritmo IPSO [176]. Como en el **Caso A**, aquí también se efectuaron 10 ejecuciones independientes con la misma cantidad de evaluaciones por ejecución a fin de poder realizar un estudio de la distribución de los valores obtenidos con mayor confiabilidad. Con ambas cantidades de ejecuciones, CPSO-shake obtuvo el mejor valor conocido $ 8,234.07, con la demanda requerida de 850 MW [199]. En la tabla 5.25 se pueden observar los valores obtenidos con CPSO-shake y con otras técnicas como un algoritmo genético [194] (GA), programación evolutiva [199] (EP), programación evolutiva mejorada [140] (IEP), método Taguchi [108] (TM), Particle Swarm Optimization modificado [138] (MPSO) y Particle Swarm Optimization integrado [176] (IPSO). Los significados de las columnas son los mismos que la tabla del **Caso A**. Como muestra la tabla 5.25, todos los métodos excepto por GA, alcanzaron el total de la potencia demandada con el mínimo costo conocido o con un valor muy cercano a éste. Al igual que en el **Caso A**, sólo IPSO y CPSO-shake pueden ser comparados justamente con respecto al tiempo computacional promedio de una ejecución. Nuevamente, CPSO-shake alcanzó la solución óptima en menor tiempo. Vale la pena destacar que EP, pese a haberse ejecutado en una computadora con menos potencia que la usada para IPSO, logró alcanzar el óptimo en un tiempo excelente. De hecho, este tiempo es mejor que el de IPSO, aunque no mejor que el de CPSO-shake.

- **Caso C**.
  Este es un problema difícil, que requiere un mayor número de evaluaciones para obtener una solución aceptable. En [176] se utilizaron 90,000 evaluaciones de la función objetivo en cada una de las 20 ejecuciones independientes, por lo cual se seleccionó esta misma cantidad de evaluaciones a fin de poder realizar una comparación justa. El mejor valor reportado es $122,190.63 [176] y por ser una función

no suave, puede existir una solución mejor. La potencia demandada es de 10,500 MW. En la tabla 5.26 se muestran los mejores valores obtenidos con CPSO-shake y un algoritmo de programación evolutiva clásico [172] (CEP), Fast Evolutionary Programming [172] (FEP), Modified Fast Evolutionary Programming [172] (MFEP), un FEP mejorado [172] (IFEP), método Taguchi [108] (TM), Particle Swarm Optimization modificado [138] (MPSO) y un Particle Swarm Optimization integrado [176] (IPSO). En la columna 1 de la tabla 5.26 se encuentran los métodos evaluados, en la columna 2 el costo medio (promediado sobre las 20 ejecuciones), la columna 3 presenta el tiempo promedio en segundos de una ejecución simple y, la última columna, el costo mínimo alcanzado ($). Todos los algoritmos comparados lograron producir el total de potencia demandada aunque los costos finales obtenidos presentan algunas variaciones. El mejor costo total fue alcanzado con CPSO-shake. Con respecto al costo computacional, nuevamente CPSO-shake es el que obtuvo la solución en menor tiempo, en especial con respecto al valor de IPSO que es con el cual es justo realizar una comparación. Nótese que el costo medio de CPSO-shake es bastante mayor que el de IPSO, lo cual deja pendiente un estudio de los valores obtenidos en cada ejecución para evaluar la robustez del algoritmo. En la tabla 5.27 se pueden observar los valores de cada una de las 40 unidades tanto en potencia como su correspondiente costo ($).

Como conclusión de la comparación indirecta de resultados, CPSO-shake obtuvo la planificación ideal para cada caso, satisfaciendo las restricciones y en un tiempo considerablemente mejor que el de los otros métodos. Debido a la no disponibilidad de los códigos fuente de las técnicas utilizadas en la comparación, se realizó un estudio estadístico de la distribución de los datos en la población de los mejores valores obtenidos para cada caso, a fin de investigar la robustez del algoritmo.

Tabla 5.24: Mejores valores obtenidos para el **Caso A**.

Método	**U1 | U2 | U3**	**Tiempo de Ejecución**	**Potencia (MW)**	**Costo Total ($)**
NM [197]	393.170 | 334.604 | 122.226	N/A	850	8,194.3561
MHNN [139]	393.800 | 333.100 | 122.300	60[a]	849.2	8,187.0000
IEP [140]	393.170 | 334.603 | 122.227	N/A	850	8,194.3561
MPSO [138]	393.170 | 334.604 | 122.226	N/A	850	8,194.3561
IPSO [176]	393.170 | 334.604 | 122.226	0,42[b]	850	8,194.3561
CPSO-shake	393.170 | 334.604 | 122.226	**0.01**[c]	850	8,194.3561

[a] Tiempo de simulación en IBM PC-386.
[b] Pentium IV 3GHz y 512MB RAM.
[c] AMD Athlon 1,24GHz con 512MB RAM.

En la figura 5.15 se puede observar que la caja de Tukey para el caso A evidencia una buena robustez del algoritmo ya que la caja no tiene altura (no existe dispersión).

Tabla 5.25: Mejores valores obtenidos para el **Caso B**.

Método	**U1 | U2 | U3**	**Tiempo de Ejecución**	**Potencia (MW)**	**Costo Total ($)**
GA [194]	300.000 | 400.000 | 150.000	16[a]	850	8,237.6000
EP [199]	300.260 | 400.000 | 149.740	0,09[b]	850	8,234.0700
IEP [140]	300.230 | 400.000 | 149.770	N/A	850	8,234.0900
TM [108]	300.270 | 400.000 | 149.730	N/A	850	8,234.0700
MPSO [138]	300.270 | 400.000 | 149.730	N/A	850	8,234.0700
IPSO [176]	300.270 | 400.000 | 149.730	0,5[c]	850	8,234.0700
CPSO-shake	300.270 | 400.000 | 149.730	**0.02**[d]	850	8,234.0700

[a] IBM Model.
[b] PC-486.
[c] Pentium IV 3GHz con 512MB RAM.
[d] AMD Athlon 1,24GHz con 512MB RAM.

Tabla 5.26: Mejores valores obtenidos para el **Caso C**.

Método	**Costo medio($)**	**Tiempo de Ejecución**	**Costo Total($)**
CEP [172]	124,793.48	1,956,96[a]	123,488.29
FEP [172]	124,119.37	1,039,16[a]	122,679.71
MFEP [172]	123,489.74	2,196,19[a]	122,647.57
IFEP [172]	123,382.00	1,167,35[a]	122,624.35
TM [108]	123,078.21	94,28[b]	122,477.78
MPSO [138]	N/A	N/A	122,252.26
IPSO [176]	122,304.70	14,56[c]	122,190.63
CPSO-shake	122,937.22	**3.36**[d]	**122,102.00**

[a] Pentium II 350MHz con 128MB RAM.
[b] Información de hardware no disponible.
[c] Pentium IV 3GHz y 512MB RAM.
[d] AMD Athlon 1,24GHz con 512MB RAM.

Tampoco se muestran colas en la distribución de datos, sino sólo la presencia de un valor alejado (outlier). El diagrama de dispersión ilustra las mismas características que la caja de Tukey. Para este caso CPSO-shake se comportó robustamente.
Para el caso B, la figura 5.16 muestra un poco de dispersión (altura en la caja) así como también un importante sesgo. No existen colas en la distribución pero sí un valor outlier. El diagrama de dispersión identifica los valores que provocan la dispersión de los datos suponiendo la mayor concentración de ellos alrededor de la mediana de la caja de Tukey. La robustez de CPSO-shake para este caso no es la ideal pero el rango de valores que incluye la mayor parte de los datos no evidencia un problema grave de desempeño.
En la figura 5.17, la caja de Tukey para el **Caso C** muestra una alta dispersión de los valores con colas en la distribución aunque no existe sesgo entre la mediana y el resto de

los datos. En el diagrama de dispersión se puede observar cómo los valores se distribuyen uniformemente en el rango, evidenciando una falla de robustez para este caso. Si bien el valor encontrado por CPSO-shake mejora el mejor valor conocido, la robustez es un aspecto que requiere más trabajo en un futuro.

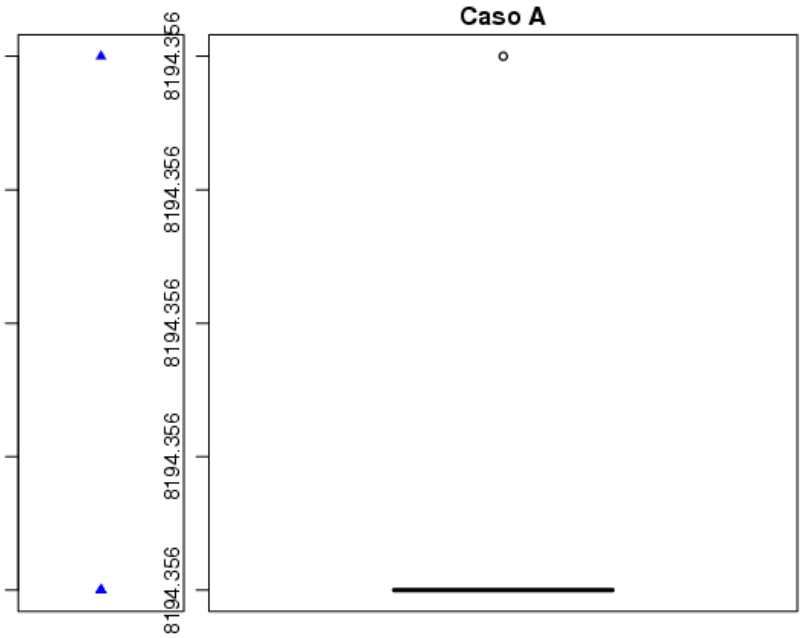


Figura 5.15: Caja de Tukey y diagrama de dispersión para el **Caso A**.

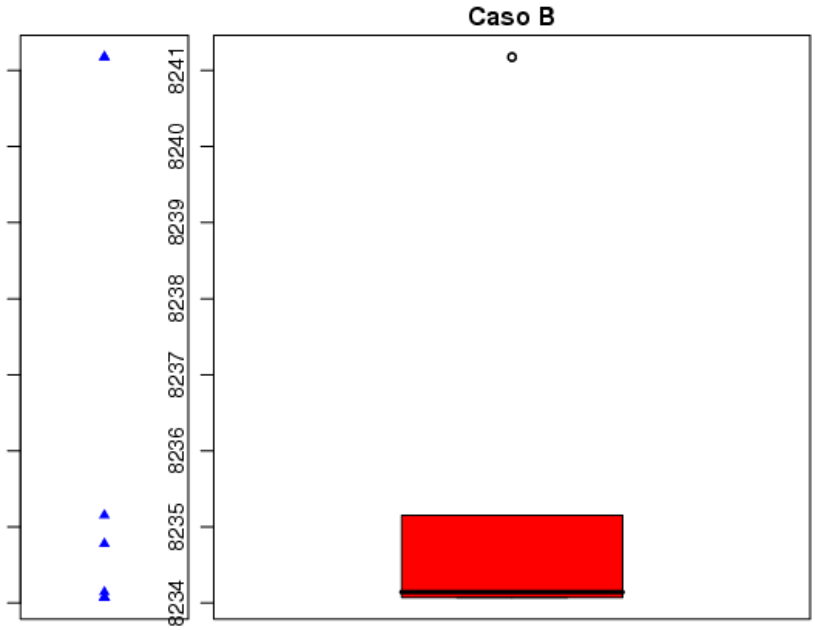


Figura 5.16: Caja de Tukey y diagrama de dispersión para el **Caso B**.

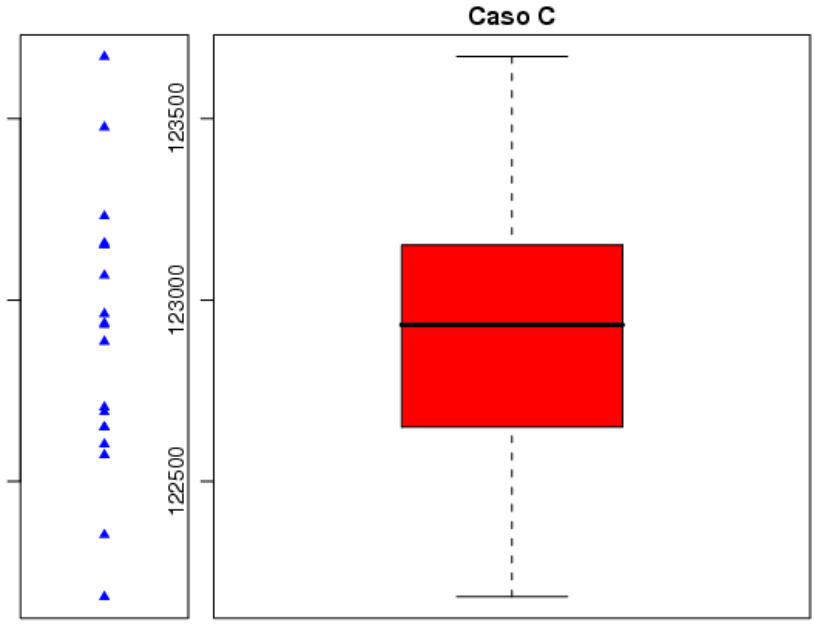


Figura 5.17: Caja de Tukey y diagrama de dispersión para el **Caso C**.

Tabla 5.27: Datos obtenidos con CPSO-shake para el **Caso C**.

Unidad	**Potencia(MW)**	**Costo($)**
1	111.350975	934.278441
2	113.000646	961.688227
3	101.183869	1,263.79287
4	179.653785	2,143.382531
5	88.748757	722.245746
6	139.838162	1,595.964151
7	259.070572	2,612.100278
8	283.384682	2,778.150071
9	284.45058	2,798.012528
10	203.433921	3,608.966312
11	169.630792	2,978.605141
12	96.759801	1,969.618297
13	305.480028	5,134.852133
14	303.767971	5,147.507701
15	392.294478	6,428.195382
16	306.205148	5,211.597378
17	491.469276	5,343.858246
18	489.208354	5,288.728751
19	512.657763	5,570.800164
20	509.974661	5,540.03484
21	524.266162	5,091.271833
22	525.414979	5,114.52581
23	532.626636	5,243.911085
24	525.047777	5,092.822089
25	522.866776	5,275.295573
26	528.35423	5,378.909508
27	10.247098	1,146.237657
28	11.710313	1,181.331381
29	11.875318	1,185.419513
30	88.762214	722.468979
31	189.635109	1,642.52283
32	161.965586	1,327.358785
33	189.49853	1,641.953942
34	166.970272	1,623.248355
35	198.070091	2,024.866572
36	168.632795	1,605.096452
37	92.831048	1,020.540419
38	89.770777	970.193241
39	106.543766	1,195.934737
40	513.346297	5,585.715229
MW & $:	**10,500**	**122,102.003178**

5.9. Conclusiones

En este capítulo se introdujo un algoritmo basado en Particle Swarm Optimization denominado CPSO-shake para resolver problemas de optimización mono-objetivo con restricciones. CPSO-shake fue utilizado en la resolución de 24 funciones con diferentes características que constituyen un benchmark comúnmente empleado en Computación Evolutiva. El desempeño del algoritmo resultó ser adecuado y comparable con el de otros métodos más sofisticados. De la comparación con otro algoritmo basado en PSO se concluyó que CPSO-shake, a pesar de ser relativamente simple, alcanza valores mejores con la misma cantidad de evaluaciones. También se resolvieron funciones de diseño de armaduras, que son problemas reales que necesitan ser optimizados en un tiempo relativamente corto y con alta calidad de resultados, al igual que el diseño de piezas y el problema de despacho de cargas eléctricas. CPSO-shake tuvo muy buen desempeño para cada uno de los casos analizados, alcanzando los mejores valores conocidos (en problemas de diseño de piezas) o mejorándolos (diseño de armaduras y problema de despacho). Para las instancias fáciles, CPSO-shake se comporta robustamente y, a medida que la complejidad de las mismas aumenta, la robustez se convierte en un aspecto que debe ser controlado. La conclusión general es que CPSO-shake resulta ser un algoritmo promisorio para resolver eficientemente problemas del mundo real, por su calidad y rapidez de respuesta.

Capítulo 6

Solución de Problemas Multiobjetivo

Un número considerable de problemas de la vida real requieren de la optimización de dos o más objetivos los cuales, en general, están en conflicto unos con otros al mismo tiempo. En este capítulo se describen algunos conceptos relacionados a la optimización multiobjetivo y se enumeran varias técnicas de resolución. Existen actualmente algoritmos PSO que resuelven eficientemente problemas multiobjetivo de mediana complejidad, algunos de los cuales están incluídos en el estado del arte en optimización multiobjetivo que se presenta en este capítulo. En la literatura especializada se pueden encontrar problemas multiobjetivo difíciles que, debido a su alta complejidad, no han sido resueltos por la mayoría de los algoritmos existentes. Esta es la razón por la cual se propone una técnica matemática hibridizada con un algoritmo Particle Swarm Optimization para la resolución de problemas multiobjetivo complejos. Los resultados obtenidos con la propuesta, demuestran que esta técnica es adecuada para resolver problemas de optimización multiobjetivo difíciles.

Contenido del Capítulo

6.1. Organización del Capítulo

El capítulo se organiza de la siguiente manera: se comienza con algunas definiciones útiles, relacionadas al problema de optimización multiobjetivo. Se realiza, luego, un repaso de las técnicas de resolución más utilizadas en el campo de la optimización multiobjetivo. Un breve estado del arte introduce las publicaciones más representativas en cuanto a la resolución de problemas con varios objetivos, utilizando PSO. Se detalla la motivación del trabajo y se describe, posteriormente, la propuesta. El capítulo finaliza con el estudio del desempeño del método presentado y las conclusiones que se desprenden del mismo.

6.2. El Problema de Optimización Multiobjetivo

En [137] se define el Problema de Optimización Multiobjetivo como: "La tarea de encontrar un vector de variables de decisión que satisfaga restricciones y optimice un vector de funciones objetivo. Esas funciones, generalmente en conflicto unas con otras, describen matemáticamente un criterio de desempeño. Por lo tanto, el término *optimizar* significa encontrar un vector solución con valores aceptables para todas las funciones objetivo."
Los problemas multiobjetivo son aquellos en los que se deben optimizar k funciones objetivo simultáneamente. El proceso de optimización puede significar la maximización de las k funciones, la minimización de las k funciones o una combinación de maximización y minimización de estas funciones.

Definición 1 [185]: ***Problema de Optimización Multiobjetivo***. Un Problema de Optimización Multiobjetivo se define como la tarea de minimizar (o maximizar) $F(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x}))$ sujeto a $g_i(\vec{x}) \leq 0 \quad i = \{1, 2, \ldots, m\}$ (restricciones de desigualdad) y $h_j(\vec{x}) = 0 \quad j = \{1, 2, \ldots, p\}$ (restricciones de igualdad) con $\vec{x} \in \Omega$. Un solución a este problema, minimiza (o maximiza) las componentes del vector $F(\vec{x})$ donde $\vec{x}$ es un vector n-dimensional de variables de decisión $\vec{x} = (x_1, x_2, \ldots, x_n)$ de algún universo Ω. Las restricciones $g_i(\vec{x}) \leq 0$ y $h_j(\vec{x}) = 0$ deben ser satisfechas al mismo tiempo que se minimiza (o maximiza) $F(\vec{x})$, y Ω contiene todos los posibles $\vec{x}$ que pueden ser usados para satisfacer la evaluación de $F(\vec{x})$.

El vector de variables de decisión puede ser continuo o discreto mientras que las k funciones pueden ser lineales o no, y continuas o discretas. La función de evaluación $F : \Omega \longrightarrow \Lambda$ es una transformación del vector de variables de decisión $\vec{x} = (x_1, x_2, \ldots, x_n)$ en un vector de respuesta $\vec{y} = (a_1, a_2, \ldots, a_k)$. La figura 6.1 ilustra la situación para el caso de $n = 2$, $m = 0$, $p = 0$ y $k = 3$.

En problemas mono-objetivo, la tarea de optimización se reduce a la obtención del óptimo que minimice (o maximice) la función objetivo. Cuando el problema es multiobjetivo el significado de óptimo debe ser redefinido, ya que la presencia de varios objetivos en conflicto ocasiona que la mejora en alguno de ellos provoque el deterioro de otros. Por lo tanto, un problema de optimización multiobjetivo consiste en encontrar el mejor compromiso (balance) entre esos objetivos. Al mejor compromiso habitualmente se lo denomina

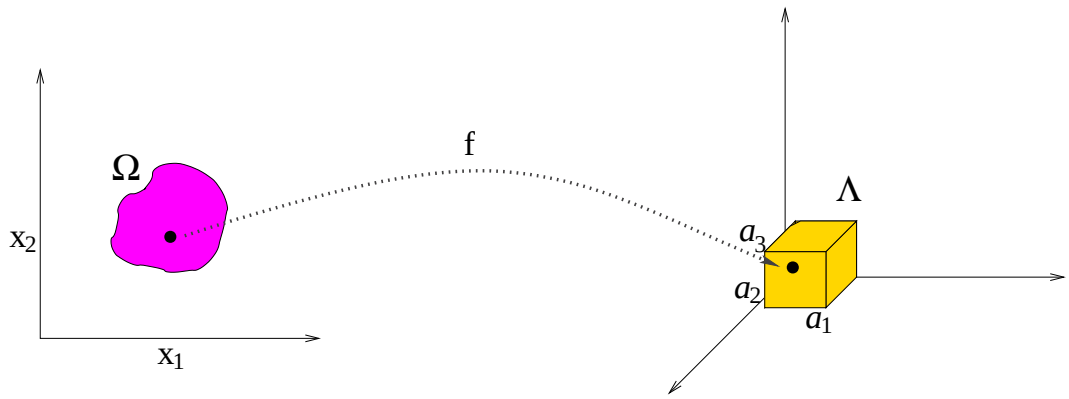


Figura 6.1: Función multiobjetivo.

óptimo de Pareto.

Definición 2 [185]: ***Optimalidad de Pareto***. Una solución $\vec{x}^* \in \Omega$ es un Óptimo de Pareto con respecto a Ω si y sólo si no existe $\vec{x} \in \Omega$, para el cual $\vec{v} = F(\vec{x}) = (f_1(\vec{x}), \ldots, f_k(\vec{x}))$ domina a $\vec{u} = F(\vec{x}^*) = (f_1(\vec{x}^*), \ldots, f_k(\vec{x}^*))$.

La definición 2 especifica que $\vec{x}^*$ es un óptimo de Pareto si no existe otro vector factible $\vec{x}$ que decremente alguna de las funciones objetivo sin causar el incremento simultáneo de otra (asumiendo minimización de todas las funciones objetivo).

Definición 3 [185]: ***Dominancia de Pareto***. Un vector $\vec{u} = (u_1, \ldots, u_k)$ se dice que *domina* a $\vec{v} = (v_1, \ldots, v_k)$, denotado por $\vec{u} \preceq \vec{v}$, si y sólo si $\vec{u}$ es parcialmente menor que $\vec{v}$, es decir, $\forall i \in \{1, \ldots, k\} : u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$.

Definición 4 [185]: ***Conjunto de Óptimos de Pareto***. Para un problema multiobjetivo determinado, $F(\vec{x})$, el Conjunto de Óptimos de Pareto, denotado por $\mathcal{P}^*$ o $\mathcal{P}_{true}$, está definido como:

$$\mathcal{P}^* := \{\vec{x} \in \Omega \mid \nexists \ \vec{x}' \in \Omega \ \ F(\vec{x}') \preceq F(\vec{x})\}. \tag{6.1}$$

Las soluciones óptimas de Pareto son aquéllas pertenecientes al espacio de decisión, cuyos vectores objetivo no pueden ser mejorados, es decir, todas las componentes de los vectores no pueden ser mejoradas simultánemente. Los vectores objetivo de las soluciones óptimas de Pareto se denominan vectores (o soluciones) *no dominados* y conforman, en conjunto, el Frente de Pareto ($\mathcal{PF}^*$).

Definición 5 [185]: ***Frente de Pareto***. Para un problema multiobjetivo determinado, $F(\vec{x})$, siendo su Conjunto de Óptimos de Pareto, $\mathcal{P}^*$, el Frente de Pareto $\mathcal{PF}^*$ o $\mathcal{PF}_{true}$, está definido como:

$$\mathcal{PF}^* := \{\vec{u} = F(\vec{x}) \mid \vec{x} \in \mathcal{P}^*\} \tag{6.2}$$

Cuando se trabaja con problemas multiobjetivo, generalmente se grafican los frentes de Pareto (vectores no dominados) generados por cada algoritmo para determinar si se ha resuelto el problema adecuadamente. Para ello se toma como referencia el verdadero frente de Pareto, es decir, el que corresponde a la mejor solución posible al problema multiobjetivo. El generado por un algoritmo deberá coincidir con éste último.

6.3. Técnicas de Resolución de Problemas Multiobjetivo

La solución de un problema de optimización multiobjetivo consta de dos etapas bien marcadas [26]: la optimización de varias funciones objetivo y el proceso de decidir cuál es el equilibrio (de objetivos) adecuado, desde el punto de vista del tomador de decisiones (*decision maker*). Una de las clasificaciones que tiene en cuenta esas dos etapas del proceso de resolución de problemas multiobjetivo, es la de Cohon y Marks.
Cohon y Marks [31] propusieron la siguiente clasificación de técnicas de resolución de problemas multiobjetivo:

- Técnicas con articulación de preferencias *a priori* (métodos no interactivos).
- Técnicas de generación, en las cuales la articulación de las preferencias se realiza *a posteriori*.
- Técnicas con articulación progresiva de preferencias, en las cuales existe interacción con el tomador de decisiones.

La clasificación de Cohon y Marks es popular debido a que toma en cuenta la forma en que se manejan los dos problemas de resolución planteados al comienzo de esta sección [72, 188].

6.3.1. Articulación de preferencias *a priori*

En este grupo se encuentran las técnicas que requieren que el tomador de decisiones exprese sus preferencias antes de efectuar la búsqueda. Algunas de las técnicas más populares dentro de este grupo se describen brevemente a continuación.

Método del Criterio Global
El objetivo de este método es la minimización de una función que define un criterio global, el cual mide cuán cerca se encuentra la solución, de la ideal. Se han propuesto diversas funciones para este método, yendo desde una métrica L_p [88] hasta variantes que penalizan las desviaciones con respecto a un objetivo que se usa como referencia [195].

Programación por Metas *(Goal Programming)*
Este es uno de los primeros métodos diseñados para resolver problemas de optimización multiobjetivo, aplicados específicamente en el campo de la industria. Fue propuesto por

Charnes y Cooper [23] e Ijiri [78] para modelos lineales. En esta técnica, el tomador de decisiones asigna metas que cada función objetivo debería cumplir. Esos valores, luego, son incorporados en el problema como restricciones adicionales, y el problema se transforma en uno de minimización de las desviaciones de los valores absolutos de cada función objetivo con respecto a su meta correspondiente.

Método Lexicográfico
En este método, el tomador de decisiones realiza una jerarquización (de mayor a menor importancia) de las funciones objetivo. Luego, la solución óptima al problema es obtenida minimizando cada función objetivo, comenzando por la de mayor importancia. El método obtiene un vector solución luego de minimizar la función objetivo más importante, y ese vector es incorporado a la siguiente función objetivo como una restricción de igualdad que se debe satisfacer. Una vez obtenida la solución para ese objetivo, se repite el proceso para la siguiente función en el orden de importancia. Para más detalles sobre esta técnica, consultar [151].

Teoría de la Utilidad Multiatributo
Von Neumann y Morgenstern [193] desarrollaron una teoría de utilidades que permite medir las preferencias individuales o de un grupo. Esta teoría presupone que un individuo puede elegir, de entre varias alternativas, la que le proporcione mayor satisfacción. Esto implica que el individuo posee conocimiento de todas las alternativas disponibles y, además, puede evaluarlas. En el método, existe asociado a cada individuo, una función de utilidad (representación matemática de la estructura de preferencias) que debe ser maximizada. Las funciones de utilidad multiatributo [81] incorporan las funciones objetivo a las estructuras de preferencias. El tomador de decisiones debe ajustar una función general en la cual se maximicen las utilidades de cada atributo.

6.3.2. Articulación de preferencias *a posteriori*

En estas técnicas, el tomador de decisiones no necesita ninguna información previa a la búsqueda de soluciones óptimas. Algunos de los métodos incluidos en esta categoría son los más antiguos en el ámbito de la optimización multiobjetivo. Los más relevantes se describen brevemente a continuación.

Combinación Lineal de Pesos
Zadeh [201], basado en la tercera condición de soluciones no-inferiores de Kuhn-Tucker, mostró que las soluciones de un problema multiobjetivo podían ser encontradas resolviendo un problema de optimización escalar, en el cual la función objetivo no es más que la suma de las funciones del problema original multiplicadas por un peso. Este método utiliza esta aproximación, aunque los coeficientes de peso no reflejan proporcionalmente la importancia relativa de los objetivos a menos que éstos estén normalizados. Los coeficientes actúan como factores que, al ir variando, localizan diferentes puntos del frente de Pareto.

Método ϵ-*constraint*
Este método minimiza una función objetivo a la vez, considerando las demás como restricciones limitadas por algún nivel ϵ_l. La selección de la primera función a optimizar y el valor de ϵ_l deben ser determinados cuidadosamente, ya que de ello depende el éxito de la técnica. A través de la variación repetida de los niveles ϵ_l, se van obteniendo las soluciones al problema. El método es también conocido como *Trade-Off*, debido a que intenta equilibrar el valor de una función con respecto a otra. Para más detalles sobre este método, consultar [111, 136].

6.3.3. Articulación progresiva de preferencias

Estas técnicas operan siguiendo tres etapas [31]: (1) encontrar una solución no dominada, (2) dependiendo de la decisión del tomador de decisiones, modificar las preferencias de los objetivos con base en la solución encontrada en la etapa anterior y, (3) repetir los pasos anteriores hasta que se cumpla la condición de parada impuesta por el tomador de decisiones. Los métodos más relevantes se describen, brevemente, a continuación.

Método de Equilibrio Probabilístico
El método presupone que, tanto las funciones como las restricciones, son probabilísticas [59]. Como primer paso del método, se encuentra una solución inicial utilizando una función objetivo sustituta (*surrogate*). Luego, a través del empleo de una función de utilidad multiatributo, se obtiene una nueva función objetivo sustituta y una nueva solución. Se comprueba que la solución sea satisfactoria para el tomador de decisiones, y se repite el proceso hasta que se alcance una solución mejor [60]. La función de utilidad multiatributo asiste al tomador de decisiones en el paso de articulación de preferencias (al comienzo del proceso), y luego, el tomador de decisiones participa interactivamente en la comparación de las preferencias con respecto al comportamiento de los atributos.

Método de Paso (*STEP*)
Este método interactivo posee bases en la articulación progresiva de preferencias. La idea subyacente es converger hacia la mejor solución utilizando no más de k pasos (siendo k el número de funciones objetivo). El método comienza con un punto ideal y, en un procedimiento de seis pasos, la va mejorando [30]. Algunas críticas indican que el método asume la no existencia de una mejor solución (de la que se encontró en los k pasos), y que no existe un equilibrio explícito entre las funciones objetivo.

Método de Resolución Secuencial
Este método fue propuesto por Monarchi et al. [126] y cuenta con una función objetivo sustituta basada en las metas y niveles de aspiración del tomador de decisiones. Las metas son condiciones impuestas al tomador de decisiones por fuerzas externas, y las aspiraciones son los logros (las funciones objetivos) que el tomador de decisiones quiere alcanzar. Las metas no cambian durante el proceso iterativo de la técnica, pero sí las aspiraciones ya que se van adecuando en cada paso del método.

6.3.4. Algoritmos Evolutivos

Aunque estos métodos de resolución no pertenecen a la clasificación previamente indicada, los algoritmos evolutivos han sido propuestos para resolver problemas de optimización multiobjetivo desde los años 1960 [158]. La primera implementación de un algoritmo evolutivo multiobjetivo fue presentada por David Schaffer en 1984 [165]: el denominado *Vector Evaluation Genetic Algorithm* (VEGA). Tanto los algoritmos evolutivos, en sus comienzos, como otros algoritmos que han emergido posteriormente, han demostrado ser técnicas adecuadas para resolver problemas de optimización multiobjetivo. El fundamento de esta aseveración es que este tipo de algoritmos puede lidiar simultánemente con un conjunto de soluciones (población). Esto permite encontrar varias soluciones del conjunto de óptimos de Pareto en una única ejecución del algoritmo, en lugar de tener que efectuar varias, como es el caso de las técnicas de programación matemática. Otra ventaja de la utilización de los algoritmos evolutivos o técnicas similares es que son menos susceptibles a la forma o continuidad del frente de Pareto, a diferencia de las técnicas de programación matemática.

Dado que el algoritmo multiobjetivo que se propone en este capítulo se basa en PSO, en la siguiente sección se realiza una breve revisión del estado del arte en torno al uso de PSO para optimización multiobjetivo.

6.4. Breve Estado del Arte en Optimización Multiobjetivo con PSO

El primer trabajo (no publicado) que extendió la heurística PSO para optimizar problemas con varios objetivos fue el de Jacqueline Moore y Richard Chapman [127]. En el algoritmo propuesto se mantiene una lista de líderes (soluciones no dominadas), la cual es generada y actualizada utilizando el concepto de dominancia de Pareto. Esta lista se emplea para seleccionar aleatoriamente soluciones que serán definidas como los valores *pbest* y *lbest*. Los autores validaron su propuesta con 2 problemas multiobjetivo de dos funciones de decisión cada uno. Los resultados concluyeron que el algoritmo propuesto obtuvo resultados comparables (para el primer problema) y mejores (para el segundo problema) con respecto a los resultados de un algoritmo genético utilizado como referencia.

En [156] se presentó un completo resumen de los algoritmos PSO más representativos del estado del arte en optimización multiobjetivo. Los autores clasificaron los algoritmos existentes (hasta la fecha de elaboración del trabajo) en seis categorías, según las características de cada método: (1) transformación a problema mono-objetivo: incluye los algoritmos que combinan todas las funciones objetivo en una única función, con el fin de realizar una optimización mono-objetivo; (2) orden lexicográfico: incluye los algoritmos en los que el usuario establece un orden de importancia en el cual cada función objetivo debe ser resuelta (optimización mono-objetivo); (3) basados en subpoblaciones: incluye los algoritmos que utilizan varias subpoblaciones (una por cada función objetivo del problema), las cuales intercambian información en el proceso de búsqueda y, posteriormente, se

combinan para obtener el conjunto de soluciones no dominadas; (4) basados en dominancia de Pareto: incluye los algoritmos que seleccionan los líderes de la población (cúmulo) utilizando el concepto de dominancia de Pareto; (5) mixtos: son aquellos algoritmos que combinan características tales como funciones ponderadas o modelos basados en agentes, con las propias del método PSO; (6) otras: incluyen los algoritmos que no pueden ser clasificados en las categorías antes indicadas.

En [13] se investiga la influencia de la selección del valor *pbest* en la optimización de problemas con varios objetivos. Los autores muestran que la elección de la partícula *pbest* tiene un impacto importante en el desempeño del algoritmo. Con base en este estudio, ellos propusieron la memorización de todas las partículas *pbest* no dominadas, de forma tal que se mantengan en un archivo externo. También propusieron 9 estrategias para seleccionar una solución no dominada del archivo, para la actualización de las mejores partículas (*pbest*): conservar la partícula más vieja (no dominada), elegir la partícula no dominada más reciente, seleccionar la partícula no dominada cuya suma de los valores de las funciones objetivo sea la mejor, seleccionar una solución del archivo en forma aleatoria, seleccionar la partícula no dominada cuya suma ponderada (se asignan pesos a cada objetivo) de los valores de las funciones objetivo sea la mejor, elegir la partícula que se encuentre más cerca de la partícula *gbest*, elegir la partícula que se encuentre más lejos de cualquier otra de la población, utilizar en la fórmula clásica de actualización de velocidades la suma de todas las partículas *pbest*, utilizar en la fórmula clásica de actualización de velocidades sólo la partícula *gbest*. Los autores utilizaron 5 problemas multiobjetivo para validar su propuesta: OKA1, OKA2, ZDT1, ZDT3 y FF. Los resultados obtenidos concluyeron que la utilización de un archivo externo mejora la calidad de las soluciones, y la selección de la estrategia para la actualización de las mejores partículas depende del problema que se está optimizando.

En [164] se hibridizó un algoritmo PSO con conceptos de la teoría de los conjuntos borrosos (*rough sets*). Los autores combinaron el buen desempeño del algoritmo PSO (demostrado en optimización de problemas mono-objetivo), con la búsqueda local que efectúan los conjuntos borrosos, para obtener una buena distribución de las soluciones en el frente de Pareto. El algoritmo propuesto se validó con 9 problemas: ZDT1, ZDT2, ZDT3, ZDT4, ZDT6, DTLZ1, DTLZ2, DTLZ3 y DTLZ4. Las conclusiones indican que la propuesta híbrida presentada obtuvo resultados competitivos con respecto a los del NSGA-II.

En [181] se presentó un algoritmo multiobjetivo basado en PSO, el cual posee: un mecanismo para mantener un conjunto de soluciones no dominadas con una buena distribución sobre el frente de Pareto, un operador de turbulencia (alteración del vuelo de las partículas) cuyo propósito es evitar la convergencia prematura a mínimos locales, un esquema para manejar las restricciones de los problemas, y un mecanismo de autoadaptación de los parámetros del algoritmo. Se utilizaron 12 problemas para validar la propuesta: CTP1,...,CTP7, ZDT1, ZDT2, ZDT3, ZDT4 y ZDT6. Los resultados obtenidos concluyeron que el desempeño del algoritmo fue bueno, en particular en los problemas con

frentes de Pareto desconectados, y que la utilización del mecanismo de autoadaptación de parámetros es una buena alternativa para facilitar el uso del algoritmo propuesto.

En [37] se realiza un estudio comparativo del desempeño de seis algoritmos PSO multiobjetivo representativos del estado del arte, los cuales se describen a continuación. (1) Non-dominated Sorting PSO [103] (NSPSO): algoritmo básico PSO al cual se le ha incorporado el mecanismo de selección de NSGA-II. Los mejores individuos se seleccionan de una población única de partículas y *pbests*, así como también de un archivo externo de buenas soluciones. El algoritmo utiliza un operador de mutación que se aplica sólo a determinadas partículas. (2) SigmaMOPSO [128]: en este algoritmo, las partículas utilizan un valor particular (*sigma*) para seleccionar a sus líderes desde un archivo externo. La propuesta emplea un operador de turbulencia para evitar convergencia prematura. (3) Optimized MOPSO [155] (OMOPSO): en esta propuesta, los líderes del cúmulo son seleccionados con base en la distancia de *crowding*, como lo hace NSGA-II. Emplea dos operadores de mutación y el concepto de dominancia ϵ para limitar las soluciones del algoritmo. En el estudio comparativo [37] se realizan las pruebas de desempeño con una versión de Optimized MOPSO sin esta última característica. (4) Another MOPSO [180] (AMOPSO): la técnica utiliza varios cúmulos en los que se ejecuta un algoritmo PSO, los cuales intercambian información relevante sobre los líderes de cada uno. El vuelo de las partículas es determinado mediante el uso del concepto de dominancia de Pareto. (5) Pareto Dominance MOPSO [3] (MOPSOpd): en esta versión de PSO, los líderes son seleccionados de un archivo externo, mediante un método basado en la dominancia de Pareto. El método puede ser aplicado de tres formas distintas (mecanismos). (6) Comprehensive Learning MOPSO [75] (MOCLPSO): al igual que MOPSOpd, este algoritmo utiliza el concepto de dominancia de Pareto para seleccionar líderes de un archivo externo. Emplea un método de distancia de *crowding* para estimar la densidad de las soluciones. Las seis propuestas de PSO fueron evaluadas con tres conjuntos de funciones: ZDT, DTLZ y WFG (21 funciones en total). Las conclusiones fueron que OMOPSO obtuvo el mejor desempeño para la mayoría de los problemas, aunque los seis algoritmos tuvieron dificultad en aproximar el frente de Pareto de 3 problemas *multi-frontales*: ZDT4, DTLZ1 y DTLZ3.

En [130], los autores propusieron un algoritmo denominado *Speed-constrained Multi-objective PSO* (SMPSO), que utiliza una estrategia para limitar la velocidad de las partículas. La idea de la estrategia es producir nuevas posiciones para aquellas partículas cuyas velocidades son muy altas. SMOPSO utiliza un operador de mutación polinomial, un factor de turbulencia y un archivo externo para almacenar las buenas soluciones encontradas. El algoritmo fue evaluado con los conjuntos de problemas ZDT y DTLZ (12 problemas en total). Las conclusiones indican que SMPSO es una técnica con muy buen desempeño en el total de problemas probados, inclusive en aquéllos que son *multi-frontales*.

6.5. Motivación del Trabajo

Los algoritmos evolutivos multiobjetivo utilizan para evaluar su desempeño, en general, problemas clásicos que están bien documentados en la literatura [26]. Estas funciones presentan diversas características en cuanto a continuidad, diferenciabilidad, convexidad, modalidad, inclusión de: restricciones, decepción y sesgos.
Los problemas numéricos sin restricciones más utilizados, son los que componen el conjunto MOP [187], denominados: MOP1, MOP2, ..., MOP7. También se emplea una gran variedad de funciones multiobjetivo con restricciones, algunas de las cuales son [26]: Binh(2), Osyczka(2), Viennet(4) y Tanaka. El conjunto de problemas ZDT (Zitzler, Deb y Thiele) [207] consta de seis diferentes funciones que fueron programadas con un generador de funciones de prueba para que incluyan ciertas características como convexidad, frentes desconectados y funciones deceptivas. Deb, Thiele, Laumanns y Zitzler [36] definieron un conjunto de funciones genéricas y escalables (DTLZ1, DTLZ2, ..., DTLZ9) que se emplean también comúnmente para validar algoritmos evolutivos multiobjetivo.
También existen, dentro de la última categoría nombrada (funciones genéricas y escalables), dos problemas propuestos por Okabe et al. [132]. Éstos fueron generados utilizando una transformación de funciones de densidad de probabilidad del espacio de decisión, en el espacio objetivo. En [132], los autores proponen dos problemas denominados OKA1 y OKA2, los cuales son complejos de resolver para cualquier algoritmo de optimización multiobjetivo, debido a que mientras más se acerca el algoritmo al frente de Pareto, la densidad de la probabilidad más disminuye. Otro conjunto de problemas genéricos y escalables, es el propuesto por Huband et al. [76]. Estos problemas se basan en un conjunto de transformaciones que se aplican secuencialmente a las variables de decisión, para agregar alguna característica particular al problema. Las funciones se denominan: WFG1, WFG2, ..., WFG9. La combinación de características en cuanto a separabilidad, modalidad, sesgo, continuidad, multidimensionalidad, y decepción provocan que estas nueve funciones sean muy complejas de resolver.
Particularmente, los dos últimos conjuntos de problemas no han sido resueltos por la mayoría de algoritmos de optimización multiobjetivo, incluso tampoco por los representativos del estado del arte. Justamente este hecho ha motivado la propuesta de un algoritmo evolutivo multiobjetivo que sea capaz de resolver eficientemente estos problemas.

6.6. El Algoritmo Propuesto: Epsilon-CPSO

El algoritmo que se propone para optimizar problemas multiobjetivo, se basa en la técnica de programación matemática ϵ-*constraint*, la cual fue hibridizada con el algoritmo CPSO-shake presentado en el capítulo anterior.
La técnica ϵ-*constraint* [64] transforma un problema de optimización multiobjetivo en varios problemas mono-objetivo con restricciones, los cuales buscan soluciones en diferentes celdas de una grilla virtual que se define previamente en el espacio objetivo.
La cantidad de trabajos publicados que utilizan la técnica ϵ-*constraint* como optimizador de funciones multiobjetivo, es bastante limitada [94, 97, 114, 150]. Ello se debe a una serie de desventajas que presenta el método. La principal desventaja es que ϵ-*constraint*

no genera un conjunto de soluciones no dominadas en una única ejecución, sólo encuentra un óptimo de Pareto por ejecución. El método, generalmente, es muy costoso porque es necesario ejecutarlo muchas veces para obtener un frente de Pareto aceptable, aún para problemas fáciles. Además, el tamaño de la grilla (cantidad de celdas definidas sobre el espacio objetivo) es fundamental para la técnica, ya que la utilización de un gran número de celdas permitirá encontrar más óptimos de Pareto, pero el tiempo de ejecución, aumentará considerablemente.
Sin embargo este método también tiene ventajas. *ϵ-constraint*, por ser una técnica de programación matemática, tiende a producir puntos de alta calidad, es decir, óptimos de Pareto o muy cercanos a ellos, aún cuando el problema sea complejo para la mayoría de los algoritmos de optimización multiobjetivo. Esto se debe a que *ϵ-constraint* busca sólo una solución en una porción del espacio objetivo, mientras que los algoritmos multiobjetivo buscan un conjunto de soluciones en el espacio objetivo completo. Además, el método obtiene soluciones eficientes en problemas no lineales, de variables enteras o mixtas [177] y, principalmente, el usuario puede seleccionar la cantidad de puntos a encontrar. Estos aspectos constituyen una justificación suficiente para utilizar este método para resolver problemas multiobjetivo, en especial, los que son complejos*.
Para acelerar el proceso de obtención del frente de Pareto del problema que se pretende resolver, en esta propuesta se aplicará el método *ϵ-constraint* para obtener una cantidad reducida de óptimos de Pareto. Para la obtención de esos puntos, se utilizará un algoritmo rápido y eficiente de optimización de problemas mono-objetivo con restricciones, como es CPSO-shake.
A continuación se describe la técnica *ϵ-constraint* y se resumen las características principales de CPSO-shake. Luego, se detalla el proceso de hibridazión de *ϵ-constraint* con CPSO-shake y, finalmente, se describe cómo aumentar la cantidad de óptimos de Pareto obtenidos con el algoritmo híbrido, para reportar un frente de Pareto adecuado.

6.6.1. El Método *ϵ-constraint*

ϵ-constraint genera un óptimo de Pareto en cada iteración del método utilizando, para ello, un optimizador mono-objetivo que maneje restricciones. Haimes et al. [64] definieron el método proponiendo:

$$\text{Minimizar} \quad f_l(\vec{x})$$

$$\text{sujeto a} \quad f_j(\vec{x}) \leq \epsilon_j \quad \text{para todo} \quad j = 1, 2, \ldots, m \quad \text{con} \quad j \neq l \quad \text{y} \quad \vec{x} \in \mathcal{S}$$

donde $l \in \{1, 2, \ldots, m\}$ y $\mathcal{S}$ es la región factible definida por las restricciones. El vector $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_m)$ define el valor máximo que puede alcanzar cada función objetivo. Variando este vector y, realizando una nueva optimización cada vez, se encontrarán nuevos óptimos. La variación de los valores de ϵ provoca que se recorra el frente de Pareto del problema. La figura 6.2 ejemplifica el funcionamiento del método *ϵ-constraint* en seis

*Landa Becerra [94] relaciona un problema complejo con el grado de dificultad que representa para un algoritmo representativo del estado del arte, como NSGA-II, el encontrar un frente de Pareto adecuado.

iteraciones. El problema consta de dos funciones objetivo y al variar seis veces el valor de ϵ, se obtienen seis óptimos de Pareto.

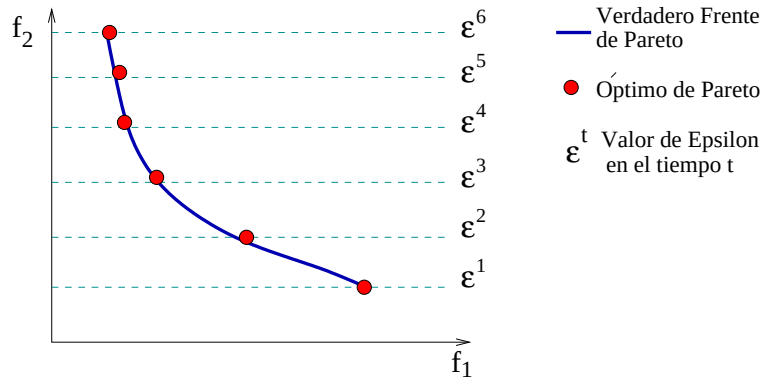


Figura 6.2: Soluciones generadas con ϵ-*constraint*.

El método ϵ-*constraint* garantiza obtener óptimos de Pareto, sólo si el optimizador mono-objetivo es capaz de encontrar el óptimo global de la función que minimiza el método (f_l). Pero está demostrado que ningún algoritmo obtiene el óptimo global de un problema no lineal, en tiempo polinomial. No obstante, el optimizador debe obtener el óptimo o un valor bastante cercano, y en un tiempo considerablemente bajo. Por tal motivo, se utiliza CPSO-shake como optimizador, resultando en una hibridización del método ϵ-*constraint* con CPSO-shake.

6.6.2. Optimizador mono-objetivo: CPSO-shake

En el Capítulo 5 se propuso el algoritmo CPSO-shake, el cual tuvo un buen desempeño en la mayoría de los casos de prueba adoptados, al encontrar soluciones cercanas o iguales al óptimo global de problemas de optimización mono-objetivo con restricciones. El costo computacional y el tiempo de ejecución requerido por CPSO-shake son relativamente bajos, lo cual favorece su utilización como optimizador en el método ϵ-*constraint*.
El Algoritmo 6.1 muestra el pseudo código de la versión CPSO-shake propuesta en el Capítulo 5. Nótese que CPSO-shake será utilizado como un procedimiento que invocará el método ϵ-*constraint*, de la siguiente manera: CPSO-shake(f_l, ϵ_j, c). Esto significa que CPSO-shake minimizará la función f_l, considerando las demás funciones del problema multiobjetivo, como restricciones. Para determinar la factibilidad de la solución (control de restricciones) se utilizará el valor ϵ_j pasado como parámetro al procedimiento, en vez de utilizar el valor estándar de 0.0001 utilizado previamente en CPSO-shake. El parámetro c indica la cantidad de ciclos que se ejecutará el procedimiento.

6.6.3. Hibridización de ϵ-*constraint* con CPSO-shake

Para que el método ϵ-*constraint* obtenga soluciones que conformen una buena aproximación del frente de Pareto, los valores de ϵ_j deben variar desde el mejor (ideal) al peor

Algoritmo 6.1 CPSO-shake

1: **Fase de Inicialización del cúmulo:**
2: **Para cada** partícula i, $i \in [1, N]$ **hacer**
3: **Para cada** dimensión d, $d \in [1, D]$ **hacer**
4: Aplicar a x_{id} un valor aleatorio en el rango $[Xmin, Xmax]$
5: Aplicar a v_{id} el valor inicial cero
6: **Fin Para**
7: **Fin Para**
8: **Fase de División del cúmulo:**
9: **Para cada** sub-población **hacer**
10: Calcular el $fitness\_x_i$
11: Aplicar a *pbest* y *lbest* el valor correspondiente
12: Aplicar a *gbest* el mejor valor de la sub-población
13: **Fin Para**
14: **Fase de Búsqueda:**
15: **Mientras** $current\_cycle < max\_cycle$ **hacer**
16: **Para cada** sub-población **hacer**
17: **Para cada** partícula i, $i \in [1, N]$ **hacer**
18: Buscar el líder en el vecindario de x_i
19: Aplicar a *lbest* el mejor valor del vecindario
20: Calcular v_i con la ecuación (4.2)
21: Calcular x_{id} con la ecuación (4.3)
22: **Fin Para**
23: Controlar rangos
24: Seleccionar $pbest_{SEL}$
25: **Si** porcentaje de infactibles > 10 **luego**
26: Aplicar mecanismo de *shake* (ec. (5.2)) al 50 % de la población
27: **Fin Si**
28: Actualizar pm
29: Mutar partículas dependiendo de pm
30: Calcular $fitness\_x_i$
31: Aplicar a *pbest* y *gbest* su correspondiente valor
32: Actualizar factor de tolerancia si corresponde
33: **Fin Para**
34: **Fin Mientras**
35: resultado = Mejor(mejorPop1,mejorPop2)
36: **Retornar** resultado

(nadir) valor de cada función objetivo. Es decir, se recorre el espacio objetivo completo de cada función realizando la búsqueda desde el vector ideal (ϵ_{ideal}) al vector nadir (ϵ_{nadir}), minimizando la función f_j.
Los vectores ideal y nadir son puntos cercanos al frente de Pareto, y deben ser provistos al método. No existe una técnica adecuada para el cálculo de estos vectores.
En particular, para el caso de problemas con dos objetivos f_1 y f_2, los vectores son puntos (2 coordenadas) en el espacio, y una forma simple de encontrarlos es: el punto ideal

se obtiene minimizando la función f_2 y luego obteniendo su valor correspondiente en el espacio de f_2 ($f_2(min(f_2))$). El punto nadir se obtiene minimizando la función f_1 y luego calculando su valor correspondiente en el espacio de f_2 ($f_2(min(f_1))$). Posteriormente, el método ϵ-*constraint* busca soluciones variando el valor de ϵ desde el punto ideal al nadir. La figura 6.3 ilustra un ejemplo para un problema con dos objetivos.

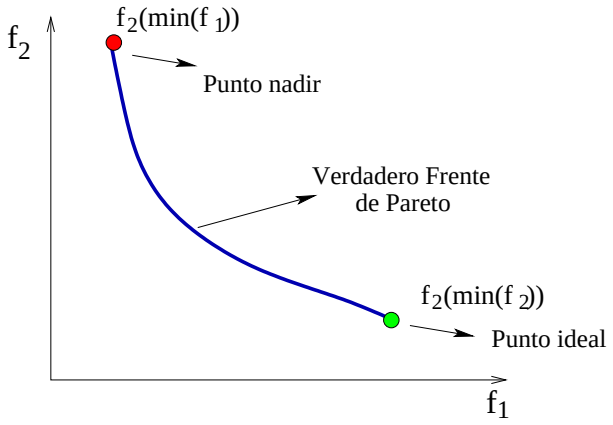


Figura 6.3: Rango de búsqueda para ϵ-*constraint* 2D.

Algoritmo 6.2 ObtenerNadir-Ideal

1: **Utilizando un algoritmo de optimización multiobjetivo hacer:**
2: obtener un conjunto de k soluciones no dominadas ND
3: **Para cada** función objetivo i **hacer**
4: NADIR[i] = mayor valor para el objetivo i de todos los puntos de ND
5: IDEAL[i] = menor valor para el objetivo i de todos los puntos de ND
6: **Fin Para**
7: **Retornar** (NADIR,IDEAL)

Para el caso de problemas con 3 o más objetivos, el ideal y nadir no son puntos, sino vectores (un valor por cada restricción f_j), y no es una tarea sencilla identificar el mejor y el peor vector de un frente de Pareto de 3 o más dimensiones.
Para el cálculo de estos vectores, se propone el procedimiento detallado en el Algoritmo 6.2. Como algoritmo de optimización multiobjetivo se utilizó uno basado en PSO [48], el cual demostró eficiencia y rapidez en la obtención de los óptimos de Pareto. Esto último es fundamental, ya que el algoritmo debe obtener buenos resultados (una buena aproximación del frente), en una cantidad muy reducida de iteraciones. Una vez que el algoritmo multiobjetivo encuentra un frente (línea 2), se busca el peor y mejor valor de cada función objetivo para conformar los vectores nadir e ideal (líneas 3 a 6). Si bien esos vectores no necesariamente son puntos pertenecientes al frente de Pareto, la idea es que sean aproximaciones que le permitan al método ϵ-*constraint*, moverse adecuadamente por el espacio objetivo.

El método ϵ-*constraint* híbrido (denominado epsilon-CPSO) se describe en el Algoritmo 6.3. En éste, un óptimo Pareto se obtiene en cada iteración del algoritmo y se almacena en el conjunto de soluciones no dominadas (línea 1), llamado Frente. Invocando al procedimiento que se describe en el Algoritmo 6.2 con el problema multiobjetivo y usando una cantidad pequeña de iteraciones c, se obtienen los vectores nadir e ideal (línea 2). El valor t es un factor de tolerancia cuyo fin es mover las coordenadas de los puntos nadir e ideal, ya que son aproximaciones y no puntos óptimos (líneas 3, 5 y 6). El valor 0.05 fue fijado con base en la sugerencia realizada en [96], al igual que el valor del factor de paso δ (línea 4). El valor del factor de paso δ depende del número máximo de puntos de Pareto que se desean obtener (Pts), y cuyo valor es un parámetro del algoritmo o un valor fijo para todo problema. En las líneas 7 a 9 se establecen los valores iniciales de ϵ utilizando los valores del vector ideal. Desde la línea 10 a la 23 se obtienen los óptimos de Pareto, controlando la búsqueda con el valor del vector nadir correspondiente a la última función objetivo (línea 10). En la línea 11 se obtiene un mínimo global, optimizando la primera función del problema multiobjetivo, y utilizando el vector ϵ como límite superior para las restricciones. El algoritmo CPSO-shake realizará el proceso de búsqueda empleando en total, c ciclos. Una vez encontrada la solución, se controla que sea una solución no dominada en el conjunto de óptimos Frente. Si así lo fuera, se agrega esa solución eliminando, si las hubiera, las soluciones dominadas por esta nueva solución (líneas 12 a 15). En la línea 16, se actualiza el valor de ϵ correspondiente a las segunda función objetivo, incrementando su valor en el factor de paso que tiene asociado. Antes de realizar la búsqueda de un punto, se controla que los valores de ϵ no hayan superado su valor nadir correspondiente (línea 18). Si así lo fuera, se reinicializa ϵ con su valor ideal (línea 19) y automáticamente se incrementa con el factor de paso, el siguiente valor de ϵ (línea 20). Con esto último se asegura la actualización de todos los valores de ϵ en un proceso en cascada. Una vez que la condición deja de cumplirse, se retorna el conjunto con los óptimos de Pareto obtenidos. El número de evaluaciones requeridas por epsilon-CPSO se calcula como Pts$\times c \times$ *particles*, siendo *particles* el tamaño de la población de CPSO-shake. Nótese que en el pseudo código se elige la primera función objetivo del problema para optimizar, y las restantes son utilizadas como restricciones. Esta selección es arbitraria, por lo que cualquiera de ellas podría ser utilizada en lugar de la primera. Por convención, en el estudio del desempeño de los algoritmos, siempre se optimiza f_1 y las restantes son utilizadas para restringir el espacio de búsqueda.

6.6.4. Mejora de la calidad del frente obtenido con el método ϵ-*constraint*

El método ϵ-*constraint* es eficiente en cuanto a la calidad de soluciones que obtiene (óptimos Pareto) pero computacionalmente costoso. Para todos los problemas resueltos, se fijó la cantidad máxima de puntos a obtener con el método híbrido, en Pts=50 para problemas con 2 funciones objetivo y Pts=100 para problemas con 3 funciones objetivo. Con esa cantidad, el costo computacional es aceptable así como la calidad del frente obtenido. El problema es que para los problemas con 2 funciones objetivo, el método ϵ-*constraint* híbrido no logra encontrar (o aproximar) más de 30 puntos del verdadero frente

Algoritmo 6.3 epsilon-CPSO

```
1: Frente = ∅
2: (NADIR,IDEAL) = ObtenerNadir-Ideal(problema MO,c)
3: t = 0.05 (NADIR-IDEAL)
4: δ = (NADIR-IDEAL)/Pts
5: NADIR = NADIR + t
6: IDEAL = IDEAL + t
7: Para cada j (2:m) hacer
8:    ε_j = IDEAL[j]
9: Fin Para
10: Mientras ε_m ≤ NADIR[m] hacer
11:    x⃗ = CPSO-shake(f_1,ε,c)
12:    Si x⃗ es no dominada en Frente luego
13:       borrar todos los dominados por x⃗ en Frente
14:       agregar x⃗ en Frente
15:    Fin Si
16:    ε_2 = ε_2 + δ[2]
17:    Para cada j (2:m − 1) hacer
18:       Si ε_j > NADIR[j] luego
19:          ε_j = IDEAL[j]
20:          ε_{j+1} = ε_{j+1} + δ[j + 1]
21:       Fin Si
22:    Fin Para
23: Fin Mientras
24: Retornar (Frente)
```

de Pareto. Para los problemas con 3 funciones objetivo, esa cantidad disminuye a un valor entre 10 y 20, produciéndose una aproximación pobre del frente de Pareto. Para sortear esta dificultad, se mantienen las cantidades máximas de 50 y 100 puntos (la prioridad es la velocidad del método ϵ-*constraint*) y luego se aplica una técnica de interpolación simple para aumentar la cantidad de puntos del frente obtenido con el Algoritmo 6.3. La técnica empleada es una interpolación cúbica por *splines* [116], la cual utiliza polinomios de menor grado (comparada a la interpolación polinómica) en determinados subintervalos. Esto último provoca que la curva obtenida con los puntos interpolados no oscile tanto y, de esa manera, reduce los errores en la aproximación obtenida. En la interpolación cúbica por *splines*, se utilizan polinomios de grado 3. Se seleccionó por ser una de las interpolaciones más utilizadas debido a que las aproximaciones obtenidas poseen un error mínimo. El conjunto de puntos interpolados es el frente de Pareto que se obtiene como solución de la propuesta de epsilon-CPSO.

6.7. Evaluación de la Propuesta

Para validar el desempeño de epsilon-CPSO, se seleccionaron problemas con dos y tres funciones objetivo. Estos problemas se caracterizan por ser complejos [26] debido a que

los algoritmos evolutivos modernos no pueden converger al verdadero frente de Pareto, aún sin restringir el número de evaluaciones.
La evaluación del desempeño de epsilon-CPSO se realizó considerando el valor de tres métricas. Éstas se seleccionaron para estudiar diferentes características de los frentes de Pareto obtenidos:

- **Cobertura (*Two Set Coverage*) (CS)**: determina la cobertura relativa de un conjunto, con respecto a otro. Fue propuesta por Zitzler et al. [207] para valorar cuantitativamente cuánto un conjunto cubre o domina a otro. Sean X e $Y \subseteq X$, dos conjuntos de vectores de decisión (frentes), CS se define formalmente como la transformación de (X, Y) al rango [0,1], según la siguiente ecuación:

$$CS(X,Y) = \frac{|\{b \in Y; \exists a \in X : a \preceq b\}|}{|Y|}$$

  Si todos los puntos en Y son dominados o son iguales a puntos de X, entonces $CS = 1$. Si ninguno de los puntos en Y es dominado o igual a algún punto de X, $CS = 0$. Nótese que lo ideal es que $CS(X,Y)$ y $CS(Y,X)$ se consideren de manera separada, ya que $CS(X,Y)$ no es equivalente a $1 - CS(Y,X)$.

- **Dispersión (*Spread*) (S)**: determina la dispersión de los puntos sobre el frente. Fue utilizada por Deb et al. en [35] para conocer cómo están distribuidas las soluciones a lo largo del frente de Pareto. Formalmente se define como:

$$S = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - d'|}{d_f + d_l + (N-1)d'}$$

  donde N es la cantidad de puntos del frente, d_i es la distancia euclidiana entre soluciones consecutivas, d' es el valor medio de todas las distancias y, d_f y d_l son las distancias euclidianas a los extremos del frente de Pareto. Un valor $S = 0$ indica la distribución ideal (dispersión perfecta).

- **Distancia Generacional Invertida (IGD)**: determina cuán lejos, en promedio, se encuentra el verdadero frente de Pareto, del frente obtenido por el algoritmo. Esta métrica evita algunos problemas que posee la métrica Distancia Generacional, en particular, en los casos en los que el frente obtenido posee pocos puntos, pero agrupados en una sola región [186]. La expresión utilizada es:

$$IGD = \frac{1}{N}\sqrt{\sum_{i=1}^{N} d_i^2}$$

  donde N es la cantidad de puntos del verdadero frente de Pareto, d_i la distancia euclidiana (medida en el espacio de las funciones objetivo) entre cada punto del verdadero frente, al punto más cercano del frente obtenido. Un valor $IGD = 0$ indica que el frente obtenido es el verdadero frente de Pareto.

Los resultados de epsilon-CPSO fueron comparados con los de NSGA-II [35] en forma directa [**], y con ϵCCDE [94] en forma indirecta.
NSGA-II fue seleccionado por ser uno de los algoritmos evolutivos más eficientes en optimización multiobjetivo. Utiliza una política elitista (siempre conserva las mejores soluciones) y su esquema de asignación consiste de un ordenamiento de soluciones, basado en la no dominancia de las mismas. Una nueva población de individuos se genera mediante el uso del operador de cruza (*crossover*) binomial, y aplicando pequeños cambios en la población mediante un operador de mutación polinomial.
ϵCCDE es una variante de la Evolución Diferencial que incorpora conceptos del motor de búsqueda de los algoritmos Culturales. Estos últimos, consisten de dos componentes principales: el espacio de población (conjunto de soluciones al problema) y el espacio de creencias, que no es más que un repositorio de información aportada por cada individuo de la población, y que pueden emplear todos. La evolución diferencial fue utilizada como técnica para modelar el espacio de población. ϵCCDE fue utilizado en esta comparación porque implementa un método *ϵ-constraint* híbrido con el algoritmo cultural antes mencionado. También utiliza un algoritmo evolutivo simple como técnica de dispersión, para aumentar la cantidad de puntos obtenidos. Dicha técnica de dispersión utiliza un algoritmo evolutivo simple que adopta operadores de cruza basados en interpoladores lineales y cuadráticos.

6.7.1. Problemas con 2 Funciones Objetivo

Para el estudio del desempeño de los algoritmos para problemas con 2 funciones objetivo, se utilizaron: OKA1, OKA2, WFG1, WFG3 y WFG5. La descripción de cada problema se encuentra en el Apéndice F.
Los tres algoritmos efectuaron 15,000 evaluaciones para obtener los frentes de OKA1, WFG3 y WFG5, mientras que efectuaron 25,000 para OKA2 y WFG1. Para todos los casos, se efectuaron 30 ejecuciones independientes y se calcularon los valores medios y desviaciones estándar de cada métrica. Estas cantidades fueron sugeridas por Landa Becerra en [94]. El objetivo, fue obtener un frente de 50 puntos con cada algoritmo.
Para obtener los resultados con el algoritmo NSGA-II, se utilizó el entorno de programación JMetal 1.0 [38], con los siguientes valores de parámetros sugeridos en [38]: selección por torneo binario, operador de cruza SBX con probabilidad 0.9, índice de distribución 20.0, probabilidad de mutación polinomial con probabilidad 1/tamaño de población, y 100 individuos en la población.
Los valores de los parámetros de ϵCCDE fueron: 5 individuos y 100 generaciones con un 10 % de la población compartida entre optimizaciones independientes. Para el optimizador mono-objetivo se emplearon 20 individuos y los parámetros de evolución diferencial fueron: F=0.7 y CR=0.5. La técnica de dispersión adoptada por ϵCCDE utilizó 5,000 iteraciones para aumentar el tamaño del frente de Pareto. Para OKA2 y WFG1 se utilizaron 150 generaciones.
epsilon-CPSO utilizó los siguientes valores de parámetros: 800 evaluaciones para la obten-

[**]Se ejecutó el algoritmo con la herramienta JMetal 1.0 [38], disponible en la web. Se agradece a los autores, por la gran utilidad de este entorno de programación para optimización evolutiva multiobjetivo.

ción de los vectores nadir e ideal (8 partículas con 100 ciclos), 14,200 evaluaciones para el optimizador mono-objetivo con restricciones (10 partículas con 1,420 ciclos) para OKA1, WFG3, WFG5. Para OKA2 y WFG1: 1,000 evaluaciones para el cálculo de los vectores auxiliares (10 partículas con 100 ciclos) y 24,000 evaluaciones en CPSO-shake con restricciones (10 partículas en 2,400 ciclos). Todos los parámetros fueron seleccionados para que el número total de evaluaciones coincida con la cantidad propuesta en [94]. Los valores de parámetros para el optimizador mono-objetivo CPSO-shake son los utilizados en el Capítulo 5, para las 24 funciones del benchmark.

Tabla 6.1: Valores medios (y desviaciones) de las métricas para OKA1 2D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.6978** (0.2200)	0.7079 (0.0630)	N/D
IGD	**0.0024** (0.0006)	0.0043 0.0019	N/D
CS(epsilon-CPSO, NSGA-II)	**0.5712** (0.0861)	-	-
CS(NSGA-II, epsilon-CPSO)	0.2356 (0.0730)	-	-
CS(ϵCCDE, NSGA-II)	0.2798 (0.2179)	-	-
CS(NSGA-II, ϵCCDE)	**0.1600** (0.1384)	-	-

Los resultados de las métricas para OKA1 se muestran en la tabla 6.1. Como se puede observar, epsilon-CPSO obtiene los mejores valores de dispersión y distancia generacional invertida, aunque los valores encontrados por NSGA-II no difieren demasiado de éstos. No se encuentran disponibles (N/D) los valores de estas métricas para ϵCCDE. Con respecto a la métrica de cobertura, el frente obtenido por epsilon-CPSO cubre mejor al frente de NSGA-II, pero el de NSGA-II cubre menos al frente de ϵCCDE (en comparación al encontrado con el algoritmo basado en PSO). La figura 6.4 ilustra un frente obtenido con epsilon-CPSO y uno obtenido con NSGA-II con respecto al verdadero Frente de Pareto. La figura 6.5 [94] muestra la comparación de los frentes producidos por ϵCCDE y NSGA-II con respecto al verdadero Frente de OKA1 [***]. Nótese que como salida de cada algoritmo, existen disponibles 30 frentes diferentes (uno por cada ejecución). A efectos de ilustrar el comportamiento de los algoritmos, sólo se muestra uno de ellos (el que posee mayor cobertura sobre el frente obtenido por NSGA-II). Comparando de forma visual los frentes obtendios por epsilon-CPSO y ϵCCDE, es notorio cómo la propuesta basada en PSO encuentra puntos del verdadero frente de Pareto en la parte inferior de la curva,

[***]Se agradece cordialmente al Dr. Ricardo Landa Becerra por permitir extraer las gráficas de los frentes de Pareto de su tesis doctoral [94], para utilizarlos en este capítulo.

mientras que ϵCCDE sólo se aproxima con algunos puntos.

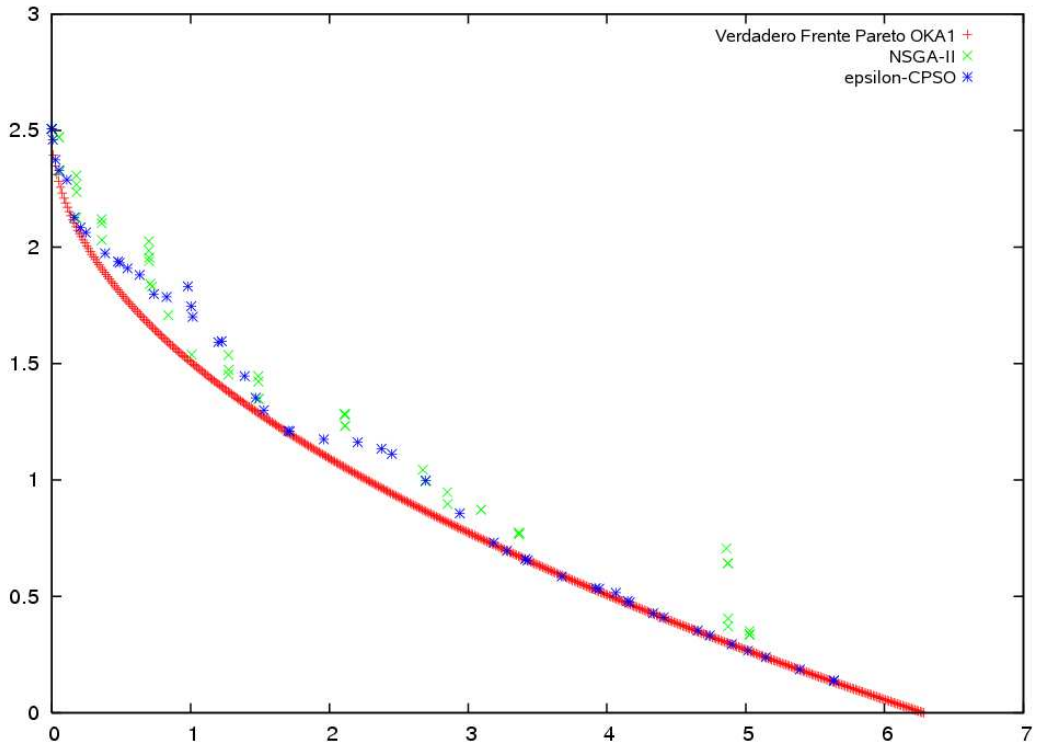


Figura 6.4: Frentes para OKA1 con dos funciones objetivo.

La tabla 6.2 muestra los valores medios (y desviaciones) de los algoritmos, para el problema OKA2. Como puede observarse, nuevamente epsilon-CPSO obtiene los mejores valores de dispersión y distancia generacional invertida, pero para este problema, la diferencia con los valores de NSGA-II es levemente notoria. Con respecto a la métrica de cobertura (dominancia del frente de NSGA-II), epsilon-CPSO obtiene un valor mejor que el que obtiene ϵCCDE. Esto significa que el frente encontrado con la propuesta basada en PSO, domina en mayor grado al frente de NSGA-II (comparado con la dominancia de ϵCCDE). Si se observa la cobertura media de NSGA-II con respecto a epsilon-CPSO y ϵCCDE, se deduce que el primero cubre menos a la propuesta basada en el algoritmo cultural, lo cual significa que existen menos puntos que dominan a los del frente de ϵCCDE. Las figuras 6.6 y 6.7 [94] ilustran los frentes obtenidos con los tres algoritmos, con respecto al verdadero frente de Pareto. El frente encontrado con epsilon-CPSO es visiblemente mejor que el de ϵCCDE ya que el primero encuentra puntos pertenecientes al verdadero frente, mientras que los puntos generados por el segundo algoritmo, se encuentran alejados del verdadero frente de OKA2. En ambas figuras es visible la dificultad que posee el algoritmo NSGA-II en aproximar el frente de Pareto para este problema.

En la tabla 6.3 se muestran los valores medios (y desviaciones) de las métricas seleccionadas, para WFG1. Los valores de dispersión y distancia generacional invertida encon-

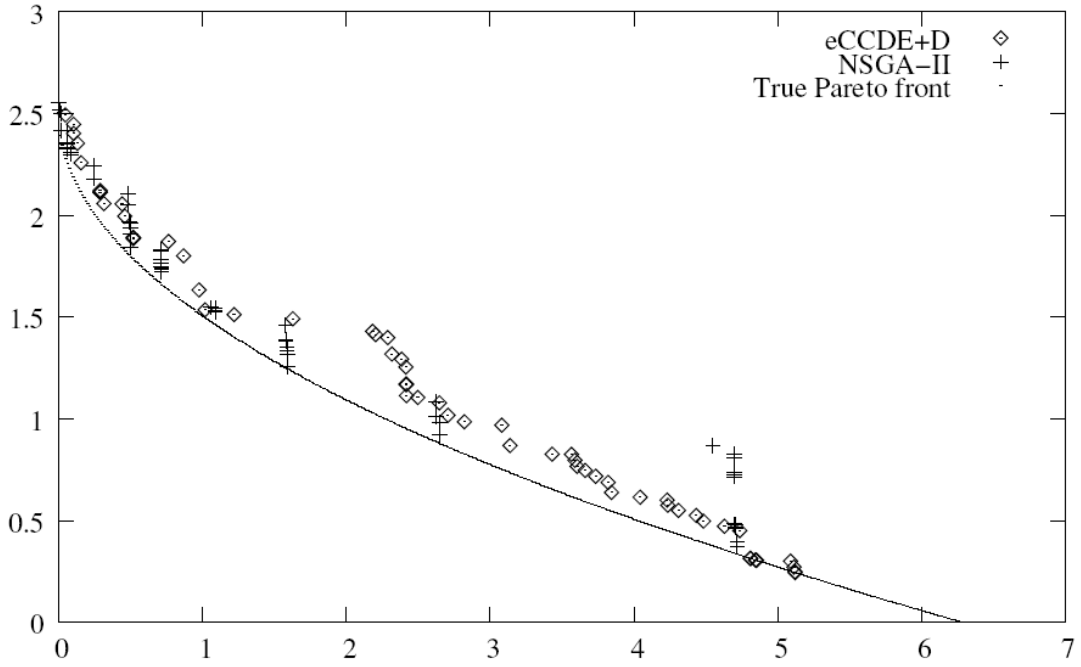


Figura 6.5: Frentes para OKA1 con dos funciones objetivo.

Tabla 6.2: Valores medios (y desviaciones) de las métricas para OKA2 2D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.9190** (0.2624)	1.1805 (0.1280)	N/D
IGD	**0.0057** (0.0025)	0.0116 0.0049	N/D
CS(epsilon-CPSO, NSGA-II)	**0.6332** (0.2980)	-	-
CS(NSGA-II, epsilon-CPSO)	0.2287 (0.1505)	-	-
CS(ϵCCDE, NSGA-II)	0.4165 (0.1593)	-	-
CS(NSGA-II, ϵCCDE)	**0.1688** (0.1062)	-	-

trados con epsilon-CPSO difieren bastante de los encontrados por NSGA-II, posiblemente por la dificultad que posee este último en aproximar el frente de Pareto. Los valores de la métrica de cobertura son extremos es decir, tanto epsilon-CPSO como ϵCCDE dominan completamente el frente obtenido con NSGA-II (CS=1), mientras que no existe cobertura de los frentes de NSGA-II con respecto a los de epsilon-CPSO y ϵCCDE (CS=0). Esto es ilustrado en las figuras 6.8 y 6.9 [94], en las cuales el frente obtenido con NSGA-II se aleja

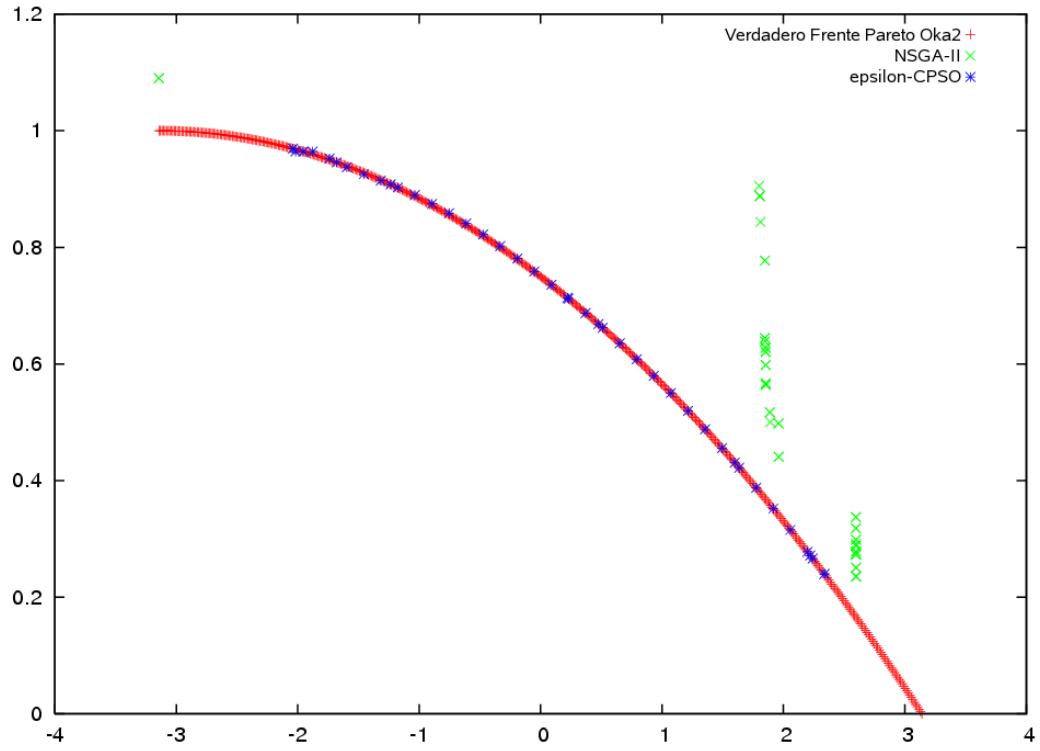


Figura 6.6: Frentes para OKA2 con dos funciones objetivo.

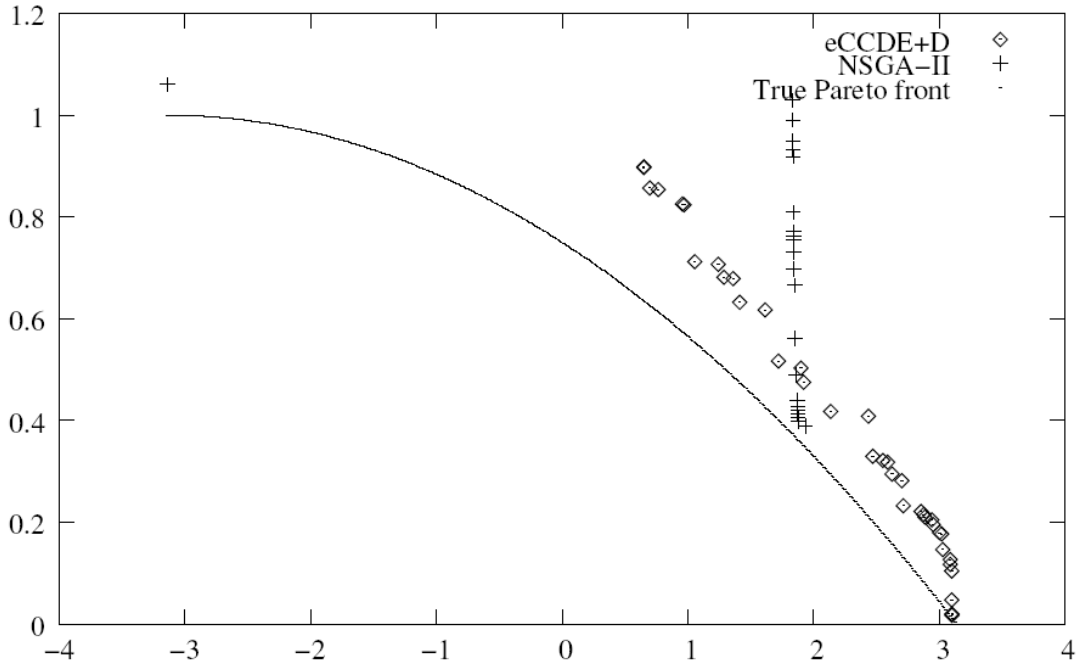


Figura 6.7: Frentes para OKA2 con dos funciones objetivo.

bastante de los frentes obtenidos con los otros dos algoritmos. Los frentes encontrados con epsilon-CPSO y ϵCCDE se asemejan en cuanto a la proximidad del frente obtenido con respecto al verdadero frente de Pareto.

Tabla 6.3: Valores medios (y desviaciones) de las métricas para WFG1 2D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.5338** (0.0365)	0.9332 (0.0841)	N/D
IGD	**0.0026** (0.0017)	0.02830 0.0542	N/D
CS(epsilon-CPSO, NSGA-II)	**1.0000** (0.0000)	-	-
CS(NSGA-II, epsilon-CPSO)	**0.0000** (1.0000)	-	-
CS(ϵCCDE, NSGA-II)	**1.0000** (0.0000)	-	-
CS(NSGA-II, ϵCCDE)	**0.0000** (1.0000)	-	-

En la tabla 6.4 se observan los valores medios (y desviaciones) para el problema WFG3 en donde, en promedio, la dispersión de epsilon-CPSO es pequeña al igual que la distancia generacional invertida (valores comparados con los de NSGA-II). Esto indica que el frente obtenido con epsilon-CPSO posee una buena cantidad de puntos y, muchos de ellos, coinciden con los del verdadero frente de Pareto. Los valores de las mismas métricas para NSGA-II son apenas superiores, con lo cual no parece ser un problema muy difícil de resolver para ese algoritmo. La cobertura del frente del algoritmo PSO con respecto al de NSGA-II, es superior que la obtenida con ϵCCDE. Con respecto a la cobertura del frente de NSGA-II, éste cubre menos al de epsilon-CPSO. Las figuras 6.10 y 6.11 [94] muestran los frentes obtenidos para WFG3. En ambas se observa que los algoritmos obtuvieron frentes aproximadamente similares.

La tabla 6.5 muestra los valores medios (y desviaciones) para el problema WFG5. La dispersión obtenida por epsilon-CPSO es inferior que la de NSGA-II, hecho que demuestra una mejor distribución de soluciones en la propuesta basada en PSO. La distancia generacional invertida para los dos algoritmos es bastante pequeña, lo que indica que muchos puntos del verdadero frente coinciden con los puntos de los frentes obtenidos. Los valores de las métricas de cobertura muestran que epsilon-CPSO posee una buena cantidad de puntos que dominan a los puntos del frente generado por NSGA-II. La cobertura de ϵCCDE sobre NSGA-II es muy cercana a cero, mientras que la opuesta es máxima. Esto último concluye que el frente de NSGA-II domina en mayor grado a las soluciones de ϵCCDE. Las figuras 6.12 y 6.13 muestran los frentes obtenidos para WFG5. En la gráfica de la figura 6.12 puede observarse que el frente generado por epsilon-CPSO se encuentra

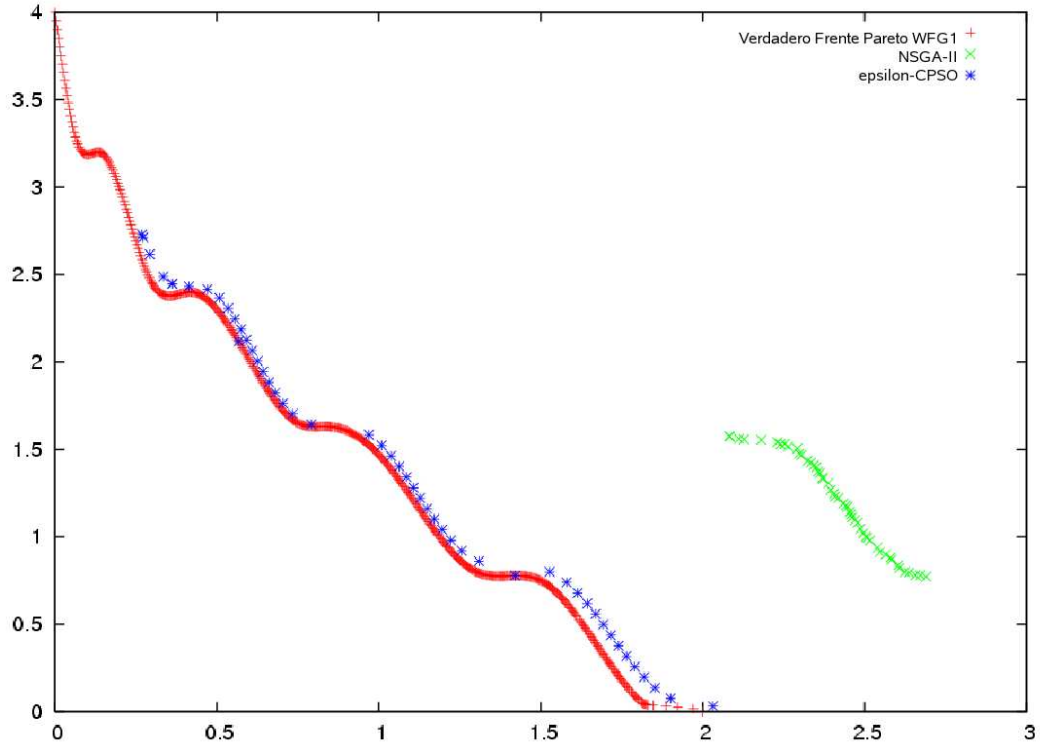


Figura 6.8: Frentes para WFG1 con dos funciones objetivo.

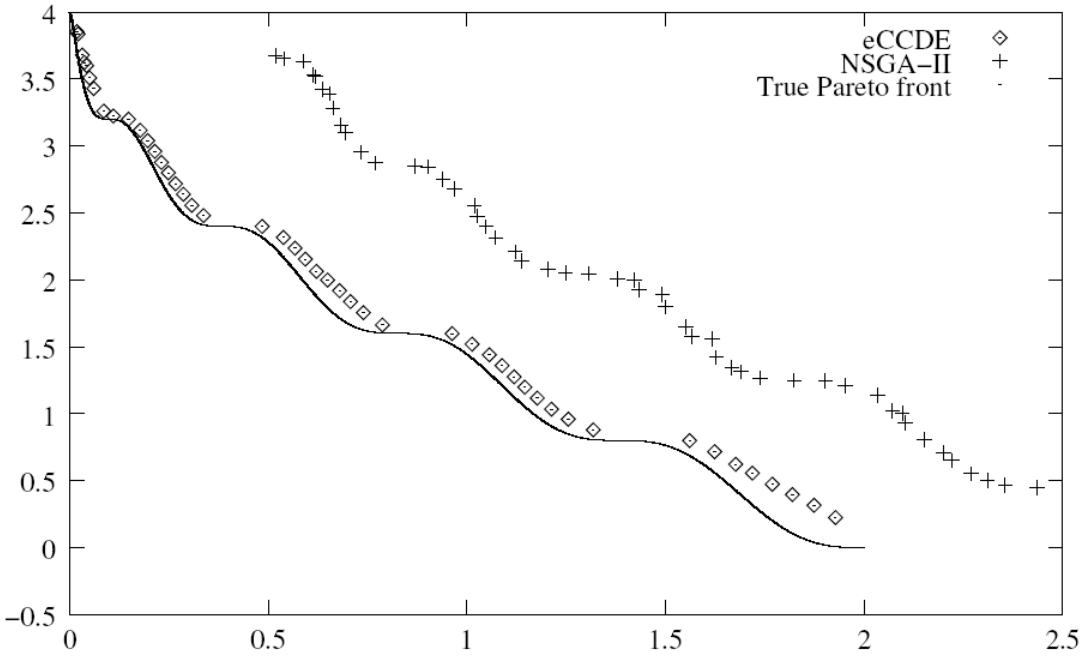


Figura 6.9: Frentes para WFG1 con dos funciones objetivo.

Tabla 6.4: Valores medios (y desviaciones) de las métricas para WFG3 2D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.2802** (0.0305)	0.4675 (0.0500)	N/D
IGD	**0.0006** (0.0000)	0.0014 0.0004	N/D
CS(epsilon-CPSO, NSGA-II)	**0.6247** (0.2465)	-	-
CS(NSGA-II, epsilon-CPSO)	**0.0612** (0.0596)	-	-
CS(ϵCCDE, NSGA-II)	0.3987 (0.2691)	-	-
CS(NSGA-II, ϵCCDE)	0.0908 (0.1368)	-	-

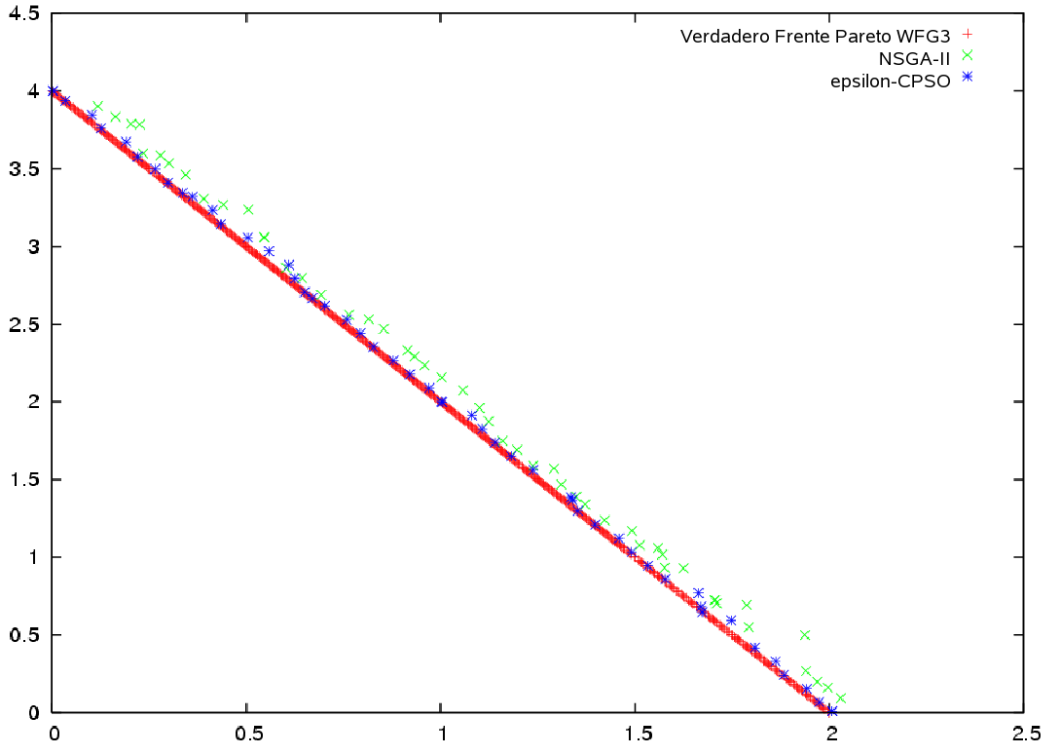


Figura 6.10: Frentes para WFG3 con dos funciones objetivo.

más próximo al verdadero frente de Pareto, que el obtenido con NSGA-II. Si se observa la figura 6.13 [94], el frente más próximo es el obtenido con NSGA-II y no con el algoritmo cultural, lo que lleva a concluir que NSGA-II tuvo mejor desempeño en este problema.

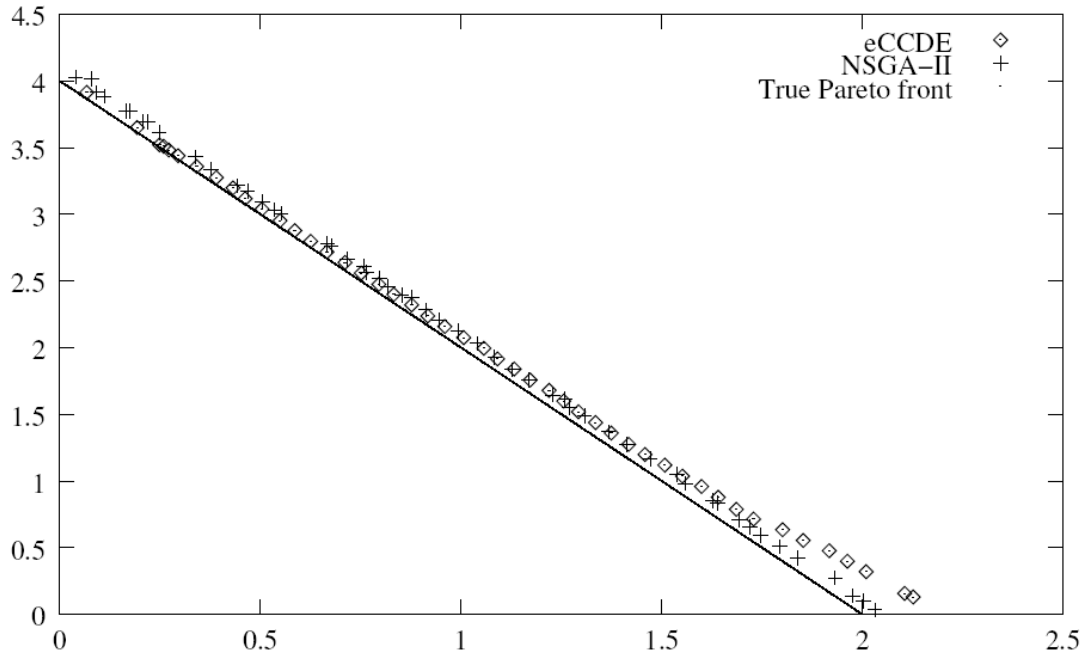


Figura 6.11: Frentes para WFG3 con dos funciones objetivo.

Tabla 6.5: Valores medios (y desviaciones) de las métricas para WFG5 2D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.5693** (0.1331)	0.8935 (0.1406)	N/D
IGD	**0.0016** (0.0001)	0.0028 0.0004	N/D
CS(epsilon-CPSO, NSGA-II)	**0.8153** (0.1540)	-	-
CS(NSGA-II, epsilon-CPSO)	**0.2426** (0.2946)	-	-
CS(ϵCCDE, NSGA-II)	0.0065 (0.0073)	-	-
CS(NSGA-II, ϵCCDE)	0.9669 (0.0461)	-	-

6.7.2. Problemas con 3 Funciones Objetivo

Para el estudio del desempeño de los algoritmos para problemas con 3 funciones objetivo, se utilizaron: WFG1, WFG3, WFG5, WFG7 y WFG9. La descripción de cada problema se encuentra en el Apéndice F.
Los tres algoritmos utilizaron 15,000 evaluaciones para obtener todos los frentes, excepto por WFG1 para el cual se emplearon 25,000. Para todos los casos, se efectuaron 30

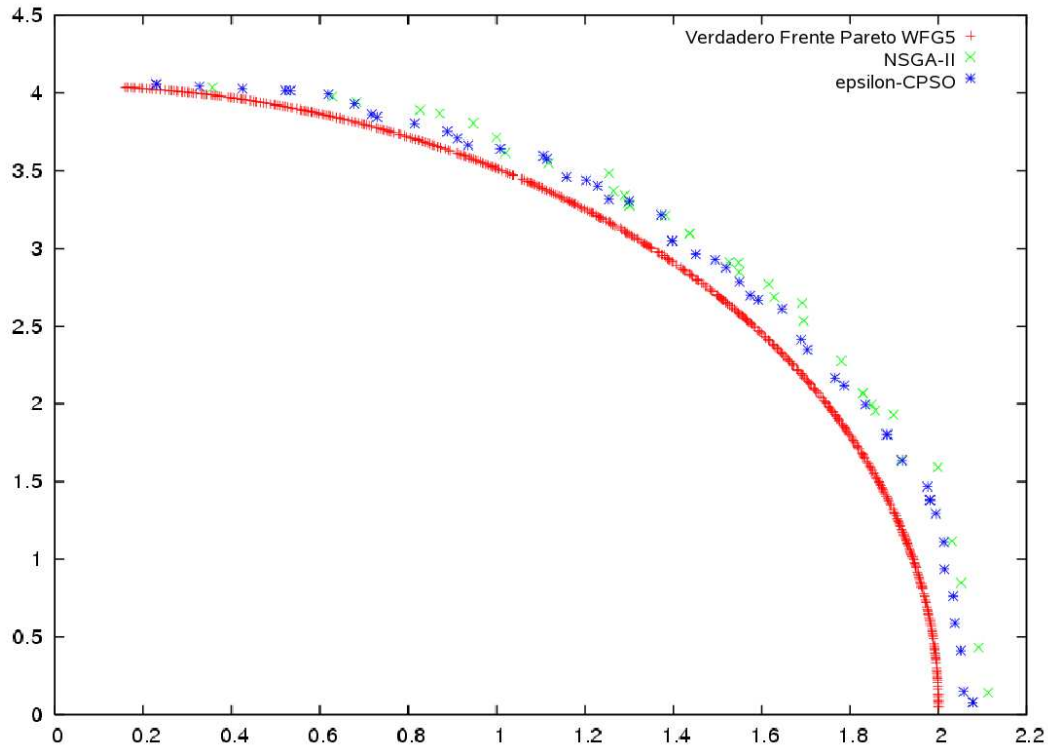


Figura 6.12: Frentes para WFG5 con dos funciones objetivo.

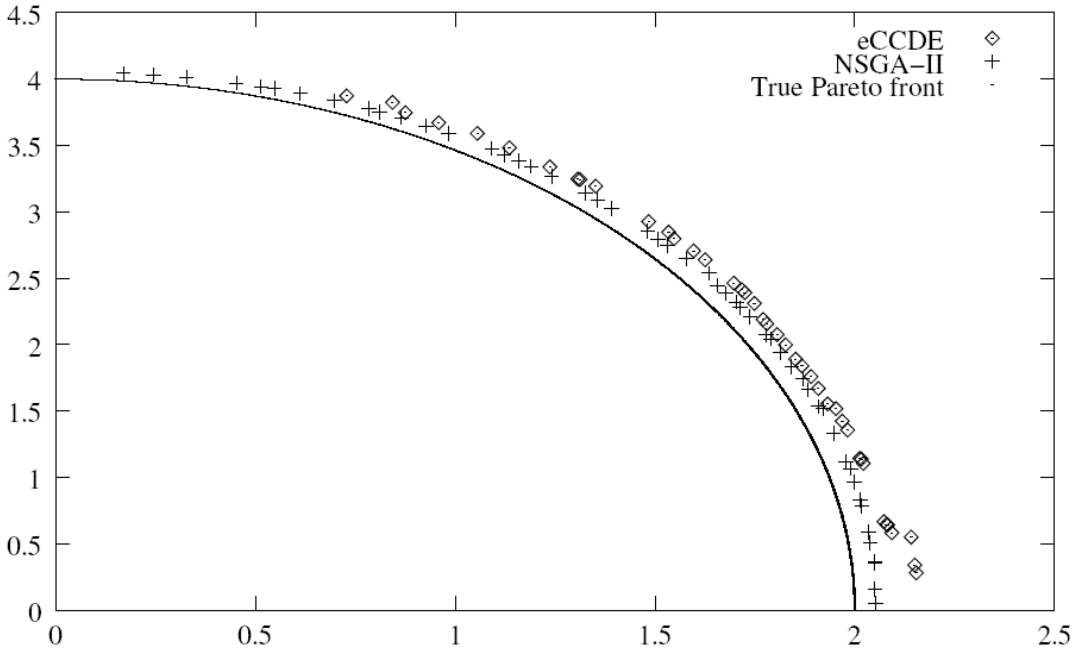


Figura 6.13: Frentes para WFG5 con dos funciones objetivo.

ejecuciones independientes y se calcularon los valores medios, y desviaciones estándar de cada métrica. Estas cantidades fueron sugeridas por Landa Becerra en [94]. El objetivo fue obtener un frente de 100 puntos con cada algoritmo, ya que al ser problemas con 3 funciones objetivo, una cantidad menor de puntos podría derivar en una aproximación no adecuada del frente de Pareto.
Para obtener los resultados con el algoritmo NSGA-II, se utilizó el entorno de programación JMetal [38] versión 1.0, en donde se sugiere utilizar los siguientes valores de parámetros: selección por torneo binario, operador de cruza SBX con probabilidad 0.9, índice de distribución 20.0, probabilidad de mutación polinomial con probabilidad 1/tamaño de población, y 100 individuos en la población.
Los valores de los parámetros de ϵCCDE fueron: 3 individuos y 70 generaciones con un 10 % de la población compartida entre optimizaciones. Para el optimizador mono-objetivo se usaron sólo 16 individuos con los siguientes parámetros para la evolución diferencial: F=0.7 y CR=0.5. La técnica de dispersión adoptada por ϵCCDE utilizó 5,000 iteraciones para aumentar el tamaño del frente de Pareto. Para WFG1 se utilizaron 104 generaciones. epsilon-CPSO utilizó los siguientes valores de parámetros: 800 evaluaciones para la obtención de los vectores nadir e ideal (8 partículas con 100 ciclos), 14,200 evaluaciones para el optimizador mono-objetivo con restricciones (10 partículas con 1,420 ciclos) para WFG3, WFG5, WFG7 y WFG9. Para WFG1: 1,000 evaluaciones para el cálculo de los vectores auxiliares (10 partículas con 100 ciclos) y 24,000 evaluaciones en CPSO-shake con restricciones (10 partículas en 2,400 ciclos). Todos los parámetros fueron seleccionados para que el total de evaluaciones coincidiera con la cantidad propuesta en [94]. Los valores de parámetros para el optimizador mono-objetivo CPSO-shake son los que se sugieren en el Capítulo 5, para las 24 funciones del benchmark.

Tabla 6.6: Valores medios (y desviaciones) de las métricas para WFG1 3D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.8183** (0.0713)	0.8583 (0.0446)	N/D
IGD	**0.3864** (0.3022)	7.4317 0.3928	N/D
CS(epsilon-CPSO, NSGA-II)	**1.0000** (0.0000)	-	-
CS(NSGA-II, epsilon-CPSO)	**0.0000** (0.000)	-	-
CS(ϵCCDE, NSGA-II)	0.8440 (0.0169)	-	-
CS(NSGA-II, ϵCCDE)	0.0000 (0.0000)	-	-

Los resultados de las métricas para WFG1 se muestran en la tabla 6.6. Como puede apreciarse, los valores medios de dispersión son bastante elevados al igual que los de la

distancia generacional invertida. No obstante, los valores encontrados por epsilon-CPSO son levemente inferiores que los de NSGA-II. Esto resume la dificultad inherente al problema en la aproximación del frente de Pareto. La métrica de cobertura del frente de epsilon-CPSO con respecto al de NSGA-II evidencia la dominancia (o igualdad) total del primero sobre el segundo. La cobertura del frente ϵCCDE con respecto al de NSGA-II es levemente inferior que la de la propuesta basada en PSO. Tanto para epsilon-CPSO como para ϵCCDE, los frentes obtenidos con NSGA-II no poseen dominancia sobre los frentes de los primeros. Esto puede ser observado gráficamente en las figuras 6.14 y 6.15 [94], en las cuales el frente de NSGA-II se aleja bastante de la aproximación obtenida con el algoritmo PSO y el cultural. Nótese también que la mayor cantidad de puntos (de los frentes obtenidos) se encuentran ubicados en la parte inferior de la curva, no pudiendo ninguno de los métodos aproximar los puntos en la parte media o superior del verdadero frente de WFG1.

Los resultados de las métricas para WFG3 se muestran en la tabla 6.7. El valor medio de dispersión obtenido con NSGA-II es levemente menor que el alcanzado con epsilon-CPSO, no siendo así con la métrica de distancia generacional invertida. Esto demuestra la dificultad de la propuesta basada en PSO en aproximar el verdadero frente, aunque los puntos que encuentra están próximos a los del verdadero frente. Esto último se deduce del valor de la distancia generacional invertida. Con repecto a la cobertura, el frente de NSGA-II domina (en una cantidad pequeña) al frente generado por epsilon-CPSO. Para el caso de ϵCCDE, el frente generado por NSGA-II domina en una cantidad mayor de puntos al frente del primero. Estos resultados se confirman en las figuras 6.16 y 6.17 [94], en las cuales el frente generado por NSGA-II se aproxima al verdadero frente de WFG3, mientras que epsilon-CPSO y ϵCCDE generan puntos alejados o muy poco aproximados al verdadero frente de Pareto.

Tabla 6.7: Valores medios (y desviaciones) de las métricas para WFG3 3D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	1.2086 (0.0748)	**0.8479** (0.0446)	N/D
IGD	**0.1811** (0.0597)	0.4107 0.3800	N/D
CS(epsilon-CPSO, NSGA-II)	0.0027 (0.0052)	-	-
CS(NSGA-II, epsilon-CPSO)	**0.0057** (0.0076)	-	-
CS(ϵCCDE, NSGA-II)	0.1273 (0.0387)	-	-
CS(NSGA-II, ϵCCDE)	**0.4146** (0.2638)	-	-

La tabla 6.8 muestra los resultados medios de las métricas (y sus correspondientes des-

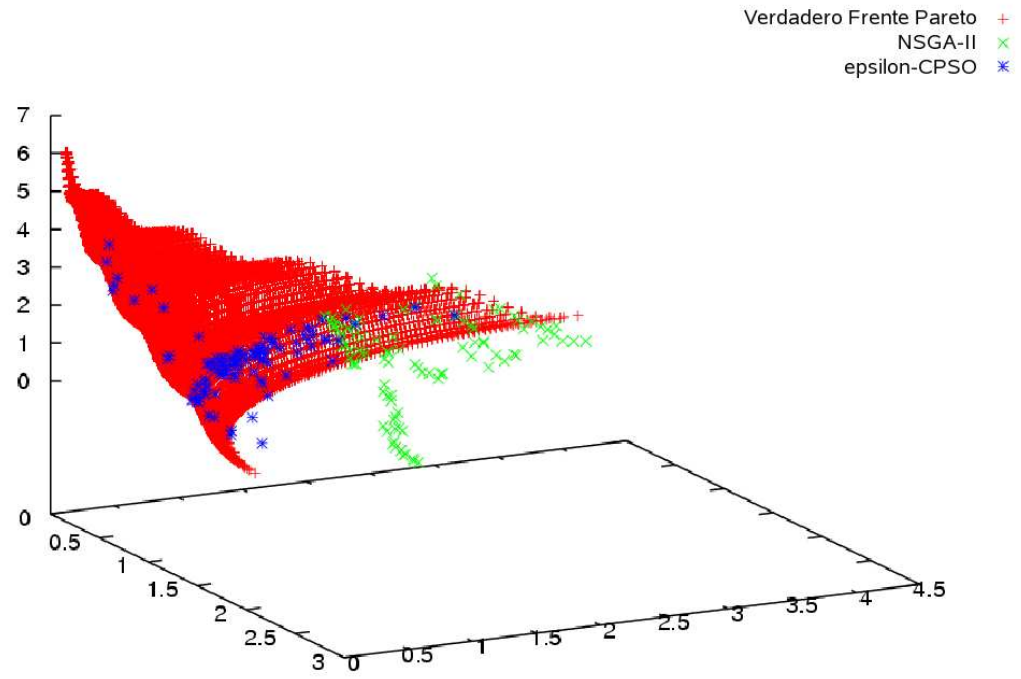


Figura 6.14: Frentes para WFG1 con tres funciones objetivo.

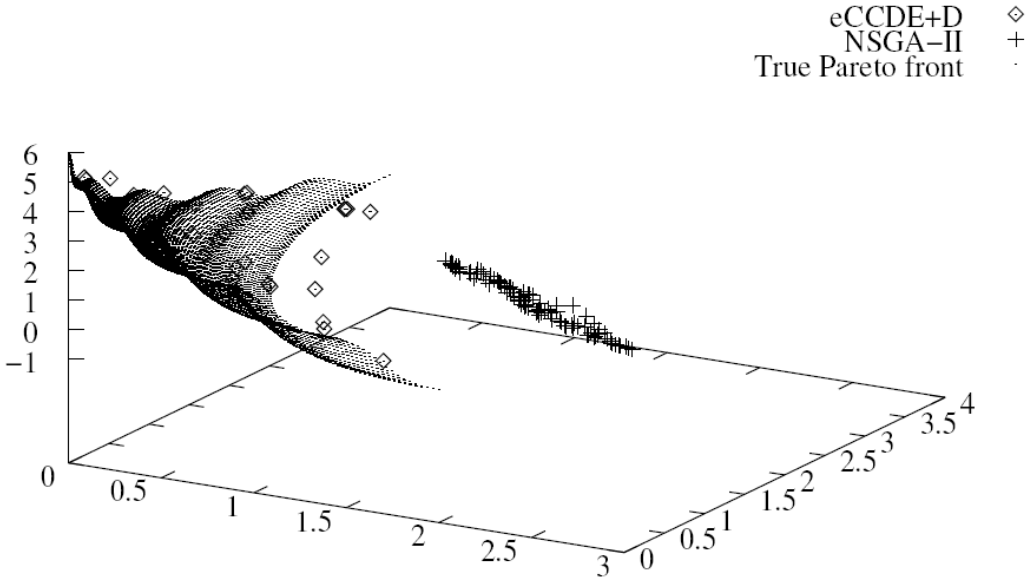


Figura 6.15: Frentes para WFG1 con tres funciones objetivo.

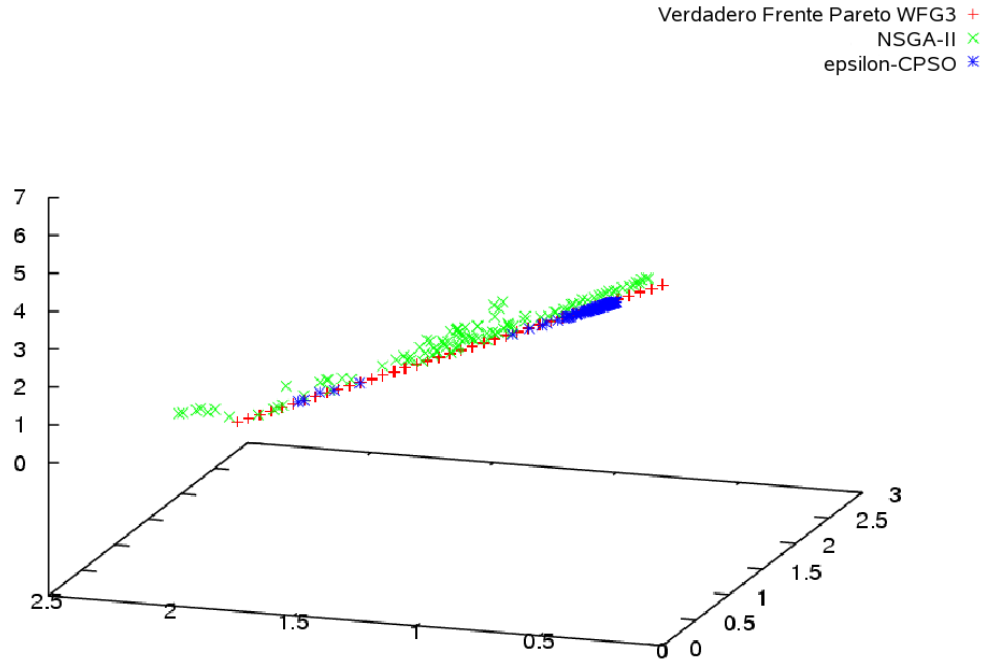


Figura 6.16: Frentes para WFG3 con tres funciones objetivo.

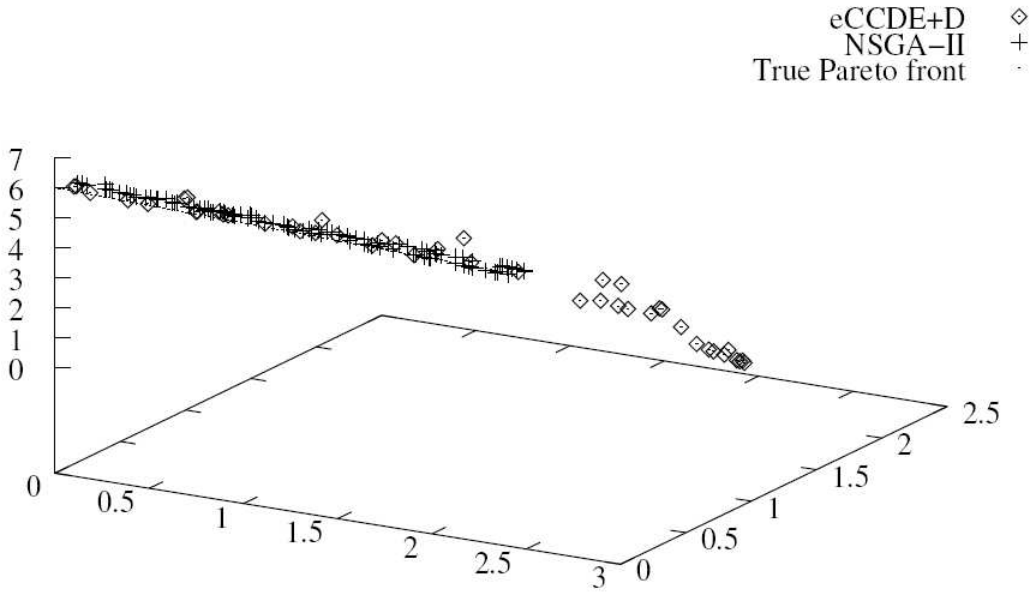


Figura 6.17: Frentes para WFG3 con tres funciones objetivo.

viaciones) para el problema WFG5. Las dispersiones medias de epsilon-CPSO y NSGA-II son similares aunque no los valores de distancia generacional invertida. De hecho, el valor de IGD obtenido con la propuesta basada en PSO es bastante menor, concluyéndose que los puntos encontrados con epsilon-CPSO pertenecen al verdadero frente de Pareto, mientras que los de NSGA-II sólo se aproximan. La cobertura del frente generado por epsilon-CPSO con respecto al frente generado por NSGA-II es mayor que la obtenida con ϵCCDE, mientras que la cobertura de NSGA-II domina en menor grado (en promedio) al frente de epsilon-CPSO. Las figuras 6.18 y 6.19 [94] ilustran los frentes obtenidos con los algoritmos, los cuales gráficamente son similares en cuanto a dispersión y cobertura del verdadero frente de Pareto.

Tabla 6.8: Valores medios (y desviaciones) de las métricas para WFG5 3D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.7559** (0.0638)	0.7813 (0.0846)	N/D
IGD	**0.0640** (0.0232)	0.4317 0.1328	N/D
CS(epsilon-CPSO, NSGA-II)	**0.0565** (0.0249)	-	-
CS(NSGA-II, epsilon-CPSO)	**0.0670** (0.0312)	-	-
CS(ϵCCDE, NSGA-II)	0.0033 (0.0027)	-	-
CS(NSGA-II, ϵCCDE)	0.2723 (0.0783)	-	-

La tabla 6.9 muestra los resultados medios de las métricas (y sus desviaciones correspondientes) para el problema WFG7. Las dispersiones medias de epsilon-CPSO y NSGA-II son similares como así también los valores de la distancia generacional invertida. Esto lleva a concluir que los algoritmos son capaces de encontrar, en promedio, frentes similares. La cobertura promedio de epsilon-CPSO con respecto a NSGA-II es levemente superior que la obtenida con ϵCCDE. En promedio, el valor de cobertura de NSGA-II es menor cuando se lo compara con ϵCCDE. Las figuras 6.20 y 6.21 [94], ilustran los frentes obtenidos por los algoritmos para WFG7. En ambas figuras puede observarse que los tres algoritmos aproximan el verdadero frente, encontrando puntos que cubren la mayor parte del verdadero frente de Pareto.

La tabla 6.10 muestra los resultados medios de las métricas (y sus desviaciones correspondientes) para el problema WFG9. Las dispersiones medias de epsilon-CPSO y NSGA-II son prácticamente similares y el valor de distancia generacional invertida obtenido con NSGA-II es levemente inferior que el de epsilon-CPSO. Esto último demuestra que en promedio, NSGA-II encuentra puntos más próximos al verdadero frente, que epsilon-CPSO.

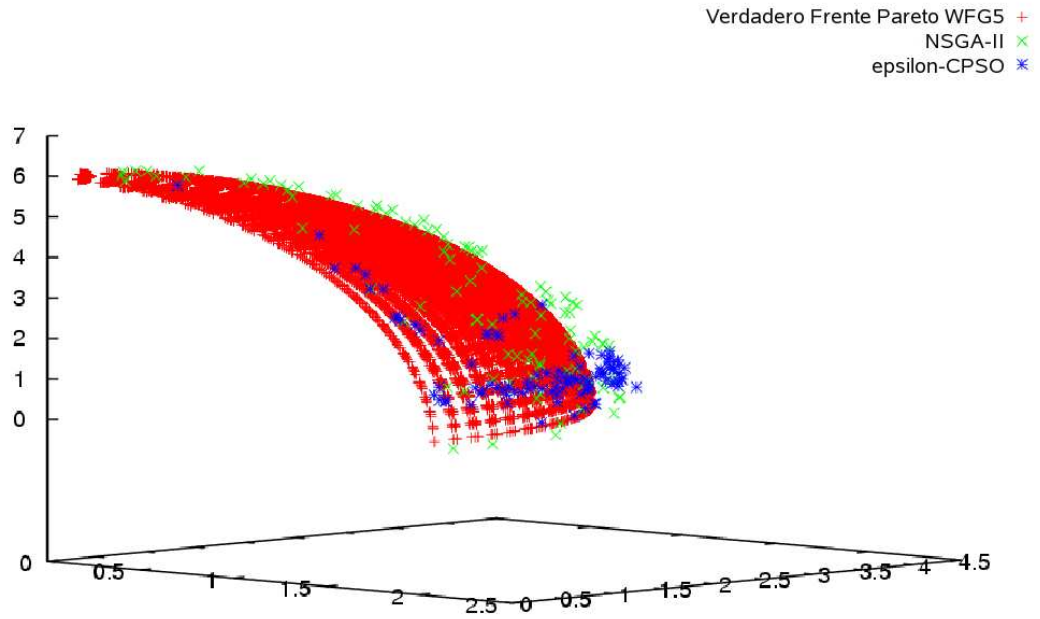


Figura 6.18: Frentes para WFG5 con tres funciones objetivo.

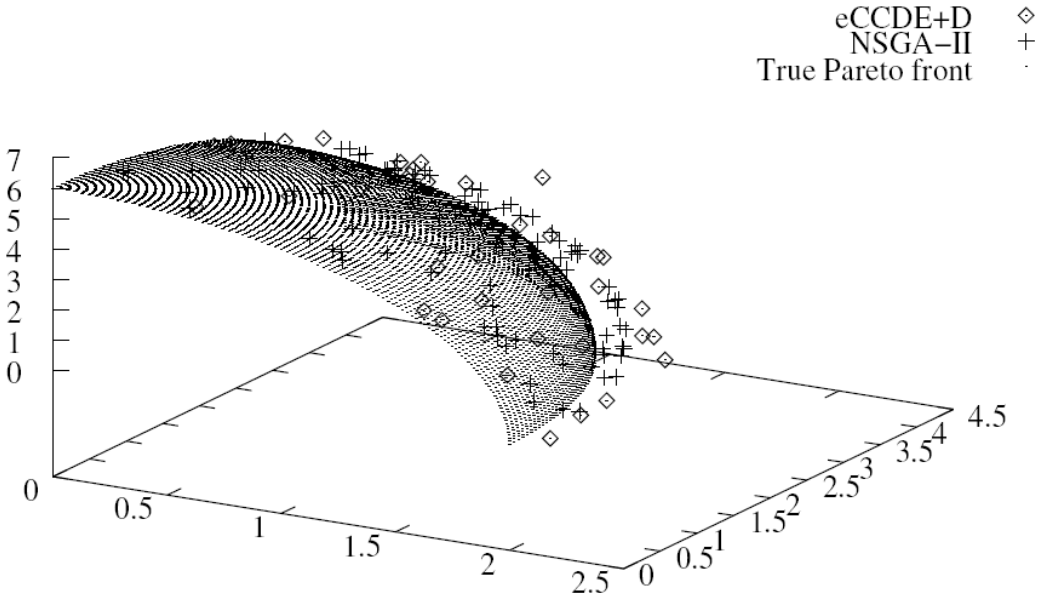


Figura 6.19: Frentes para WFG5 con tres funciones objetivo.

Tabla 6.9: Valores medios (y desviaciones) de las métricas para WFG7 3D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.8183** (0.0490)	**0.8188** (0.0816)	N/D
IGD	**0.0147** (0.0022)	0.0497 0.0121	N/D
CS(epsilon-CPSO, NSGA-II)	**0.1026** (0.0525)	-	-
CS(NSGA-II, epsilon-CPSO)	0.2742 (0.0601)	-	-
CS(ϵCCDE, NSGA-II)	0.0033 (0.0027)	-	-
CS(NSGA-II, ϵCCDE)	**0.1683** (0.0872)	-	-

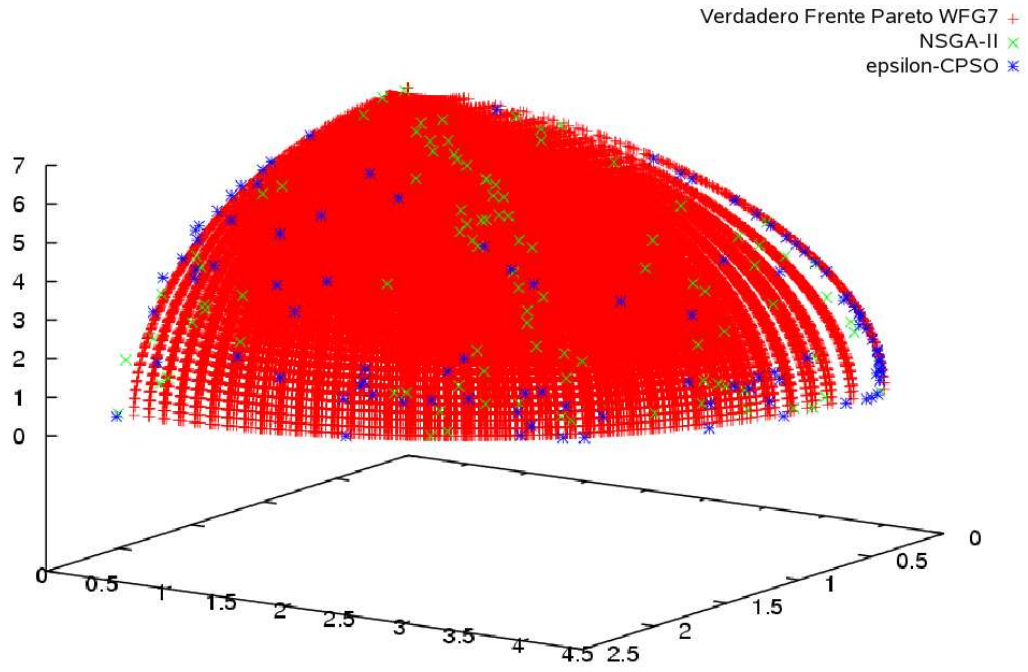


Figura 6.20: Frentes para WFG7 con tres funciones objetivo.

La cobertura promedio de epsilon-CPSO con respecto a NSGA-II es superior que la obtenida con ϵCCDE. En promedio, el valor de cobertura de NSGA-II es menor cuando se lo

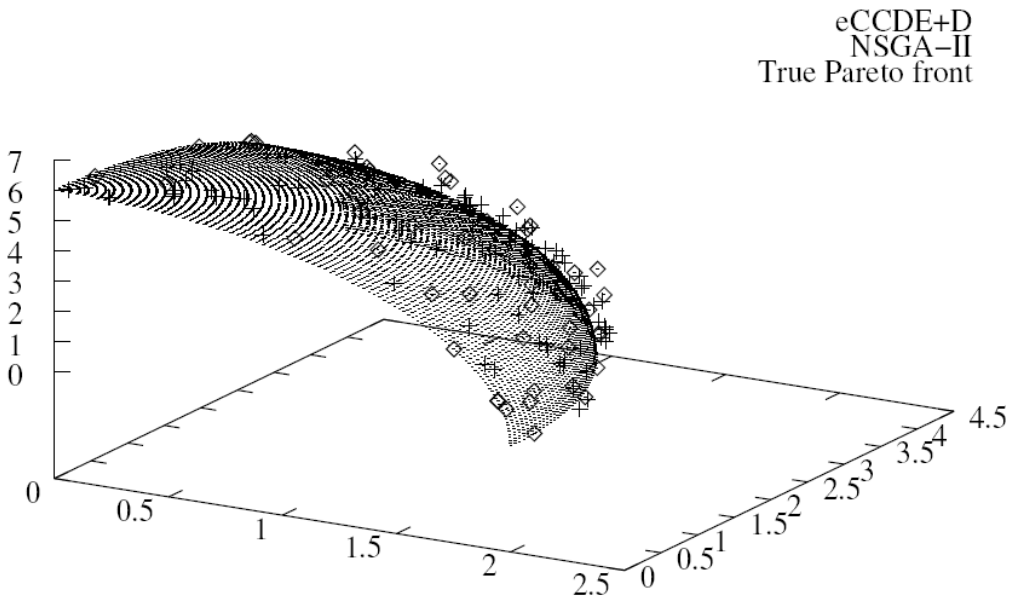


Figura 6.21: Frentes para WFG7 con tres funciones objetivo.

compara con epsilon-CPSO. Esto concluye que NSGA-II domina (cubre) menos al frente de la propuesta basada en PSO (comparado con la misma medida y el algoritmo cultural). Las figuras 6.22 y 6.23 [94], ilustran los frentes obtenidos por los algoritmos para WFG9. En ambas figuras puede observarse que los tres algoritmos aproximan el verdadero frente, encontrando puntos que cubren la mayor parte del verdadero frente de Pareto.

Tabla 6.10: Valores medios (y desviaciones) de las métricas para WFG9 3D.

Métrica	**epsilon-CPSO**	**NSGA-II**	**ϵCCDE**
S	**0.5986** (0.0438)	0.6665 (0.0416)	N/D
IGD	0.0085 (0.0002)	**0.0046** 0.0003	N/D
CS(epsilon-CPSO, NSGA-II)	**0.8665** (0.0382)	-	-
CS(NSGA-II, epsilon-CPSO)	**0.0168** (0.0681)	-	-
CS(ϵCCDE, NSGA-II)	0.6896 (0.2014)	-	-
CS(NSGA-II, ϵCCDE)	0.0543 (0.0539)	-	-

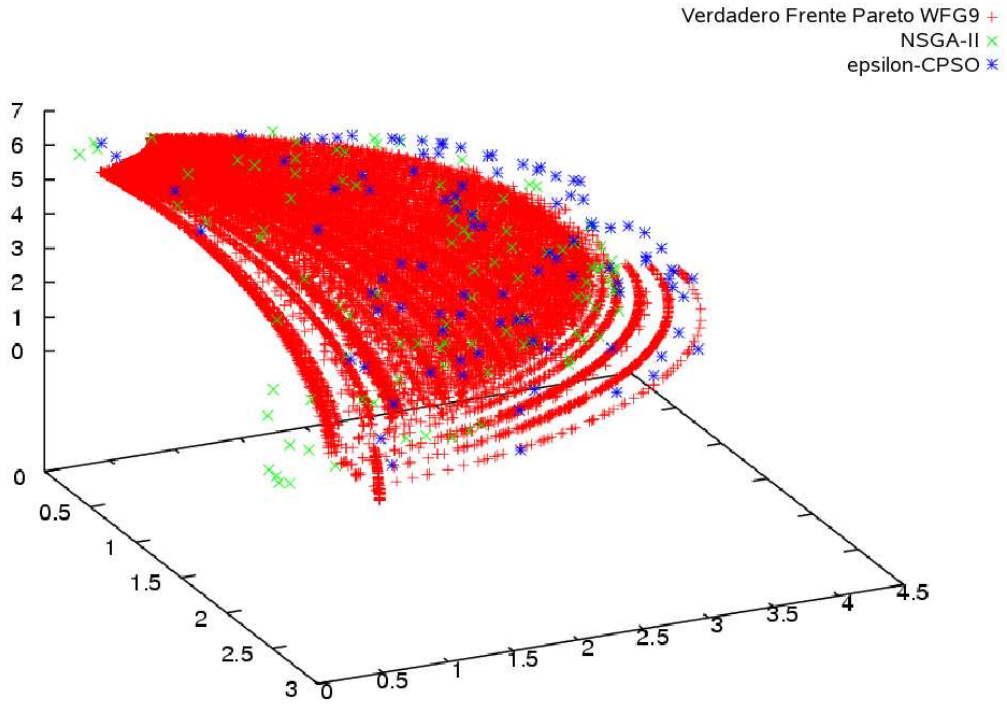


Figura 6.22: Frentes para WFG9 con tres funciones objetivo.

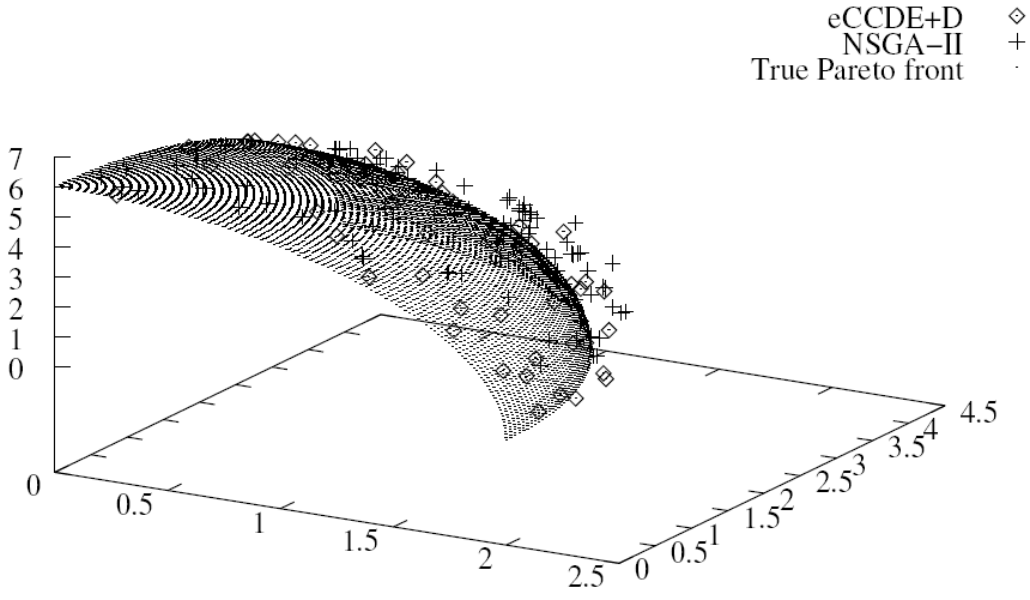


Figura 6.23: Frentes para WFG9 con tres funciones objetivo.

6.8. Conclusiones

La propuesta para problemas multiobjetivo con restricciones denominada epsilon-CPSO, fue evaluada con problemas de 2 y 3 funciones. Para todos los casos, excepto por WFG3 con tres funciones objetivo, epsilon-CPSO demostró obtener buenas aproximaciones de los frentes de Pareto con mínima dispersión y distancia generacional (al menos con respecto a los resultados de NSGA-II). En la mayoría de los problemas utilizados en esta evaluación, epsilon-CPSO obtiene en promedio, un frente que domina a uno obtenido con NSGA-II, hecho que no sucede con el algoritmo cultural ϵCCDE utilizado en este estudio. Se concluye, luego, en el buen desempeño del método híbrido epsilon-CPSO para resolver problemas de optimización multiobjetivo complejos.

Capítulo 7

Análisis de parámetros para PSO

En este capítulo se describen algunos métodos utilizados para la determinación de un conjunto de parámetros adecuados para diferentes versiones del algoritmo PSO. Se describe un estudio de los parámetros para la optimización de funciones sin y con restricciones, incluyendo la optimización de funciones de ingeniería estructural, el cual es un problema real y difícil de resolver.

Contenido del Capítulo

7.1. Introducción al análisis de parámetros

Históricamente, la determinación de los parámetros de cualquier algoritmo de optimización ha sido realizada empíricamente dependiendo del conjunto de funciones que se desea resolver. El término "empírico" se refiere a un proceso repetitivo de ejecución de un algoritmo con diferentes valores de parámetros, variándolos de manera sistemática, buscando cubrir los rangos de cada uno. Se emplean varias funciones de prueba y luego, a partir de los resultados obtenidos, se seleccionan aquellos valores para los cuales el algoritmo arrojó los mejores resultados. Aunque esto ha sido muy utilizado, hoy en día este procedimiento es considerado inaceptable. La justificación es simple: no se trata de programar un algoritmo y evaluar su desempeño con un conjunto de funciones para las cuales "se deben utilizar" ciertos valores de parámetros, sino que se trata de programar un algoritmo que utiliza ciertos parámetros y funciona bien para varios tipos de problemas. El proceso de optimización tal y como se conocía hasta ahora: "*programación del algoritmo-selección de funciones-elección de parámetros con los que mejores resultados se obtiene* ha cambiado a *programación del algoritmo-elección de parámetros con los que mejores resultados se obtiene en general-optimización de funciones*". El ajuste (*tuning*) como comúnmente se conoce al proceso de fijar los parámetros de un algoritmo, no es una tarea sencilla ni directa. Más bien es una parte del proceso de optimización que resulta casi tan importante como la programación misma del algoritmo.
En [173] se diferencian los tipos de parámetros según los valores que éstos pueden tomar. Los parámetros *numéricos* son aquellos cuyo dominio es un conjunto infinito de números reales, continuos o discretos. Los parámetros *simbólicos* son aquellos cuyos valores dependen de un conjunto finito de posibilidades que se especifican por enumeración. Ejemplo de los primeros son el tamaño de población y ejemplo de los segundos son el tipo de vecindario (topología) de un algoritmo PSO: anillo, rueda, arcos aleatorios o malla.
Formalmente, el problema de encontrar valores adecuados para los parámetros de un algoritmo se define [173] como un problema de búsqueda compuesto de parámetros simbólicos $q_1, \cdots, q_m$ con dominios $Q_1, \cdots, Q_m$ y parámetros numéricos $r_1, \cdots, r_n$ con dominios $R_1, \cdots, R_n$, en el espacio de parámetro $S = Q_1 \times Q_2 \times \cdots \times Q_m \times R_1 \times R_2 \times \cdots \times R_n$. Las soluciones al problema serán vectores de parámetros con máxima utilidad, siendo la utilidad de un vector $\vec{p} \in S$ el desempeño del algoritmo que está siendo analizado.

Existen muchas maneras de determinar el conjunto adecuado de parámetros de un algoritmo, las cuales dependen del conocimiento que se posea de técnicas estadísticas de diseño, de la cantidad de parámetros del algoritmo y del tiempo de estudio del que se disponga. Una de ellas consiste en adoptar un conjunto de herramientas de análisis de experimentos para estudiar la variabilidad de las respuestas del algoritmo frente a los diferentes conjuntos de parámetros. Generalmente se pueden distinguir dos grandes grupos en los que estas técnicas se pueden clasificar: (1) cuando la muestra utilizada se conforma de valores extremos, o (2) cuando los datos se corresponden con valores del interior del espacio de diseño.
La metodología estadística de *Diseño de Experimentos*, o DOE por sus siglas en inglés, emplea (1) y fue utilizada primeramente en optimización en las áreas de agricultura e industria aunque para ser empleada en optimización computacional estocástica fue adaptada

con la utilización de la pseudoaleatoriedad [50], es decir, utilizando números pseudoaleatorios*. El *Diseño y Análisis de Experimentos Computacionales*, o DACE por sus siglas en inglés, utiliza (2) y constituye un complemento a DOE.
El *diseño factorial* es una técnica clásica de DOE que podría ser utilizada para el estudio de parámetros de algoritmos, y que es mucho más eficiente que el muy utilizado *one-factor-at-a-time*, el cual varía un parámetro a la vez dejando los demás en valores constantes. Este último no es recomendado porque no se pueden detectar interacciones entre los diferentes parámetros del algoritmo. Por esto se sugiere la utilización de los diseños factoriales, los cuales van variando los valores de los parámetros del algoritmo todos a la vez. También es posible utilizar herramientas gráficas clásicas de DOE como las Cajas de Tukey (*Box Plots*) para determinar características de los resultados obtenidos con los distintos conjuntos de parámetros, algo que no se puede deteminar con el diseño factorial. Las cajas de Tukey fueron introducidas en el Capítulo 5 (sección 5.6) como herramientas gráficas útiles para mostrar la distribución que tienen los datos de una muestra. Para la construcción se utilizan 5 importantes medidas descriptivas: la mediana (tendencia central), el primer cuartil (el 25 % de los datos), tercer cuartil (el 75 % de los datos), valor máximo y valor mínimo (de la muestra). Las cajas de Tukey, además de representar visualmente las 5 medidas antes descriptas, informa sobre la tendencia central, la dispersión y simetría de los datos.

Los diseños factoriales utilizados generalmente en DOE, ubican la muestra de datos en los extremos del espacio de diseño, quedando el interior del mismo sin explorar. Esto se debe a que estas técnicas se basan en un modelo aproximado del problema, más un término de error. En DOE siempre se pretende minimizar la influencia del error en el modelo y esto se logra cuando la muestra se ubica en los límites del espacio de diseño y no en el interior del mismo.
Esta falla de DOE es subsanada por los métodos DACE que fueron pensados para experimentos computacionales determinísticos, los cuales no poseen errores. En DACE se presupone que tanto en los extremos, como en el interior del espacio de diseño, pueden encontrarse características importantes del verdadero modelo. Por eso es que los métodos DACE cumplen con el diseño cobertor del espacio (*space-filling*). Uno de los métodos que cumplen esta característica es el *muestreo de Monte Carlo* (con algunas modificaciones [120]) que ubica aleatoriamente los puntos de la muestra en el interior del espacio de diseño (algunas partes del espacio pueden no ser cubiertas). Una variante de Monte Carlo es la que divide ese espacio en d subintervalos de igual probabilidad y coloca los puntos de la muestra en los 2^d subintervalos. Un diseño aún mejor al de Monte Carlo es el *diseño de Hipercubos Latinos* [115] (LHD por sus siglas en inglés). LHD genera puntos aleatorios para la muestra y generalmente es superior a Monte Carlo por su flexibilidad en la selección del número de puntos a generar. LHD no necesariamente debe ser un diseño

*Los procesos involucrados en las observaciones reales como los de agricultura e industria son realmente sucesos aleatorios, pero sistemas determinísticos como las computadoras, utilizan otra clase de aleatoriedad, la cual debe ser "generada" a través de un valor "semilla". Sin embargo, si se usa la misma semilla, la secuencia generada será siempre la misma. Por ello no se habla de aleatoriedad sino de pseudoaleatoriedad cuando éstos son generados por una computadora.

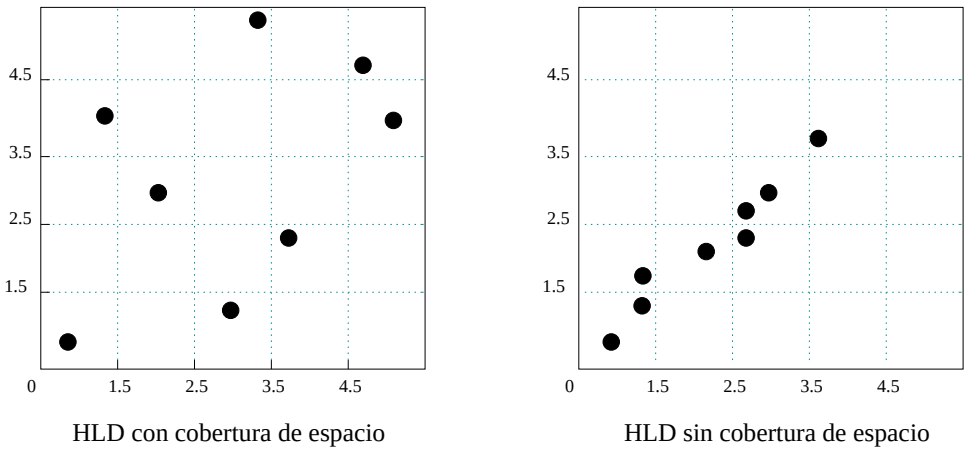


Figura 7.1: Dos instancias de diseño de hipercubos latinos.

cobertor del espacio aunque siempre es recomendable generar uno que lo sea. La figura 7.1 muestra dos instancias de LHD con ocho puntos de diseño en dos dimensiones, una con cobertura del espacio y otra sin ella.

Utilizando como argumento las propiedades de cada uno de los modelos, DACE y DOE, se presenta en la próxima sección un estudio completo de los parámetros de tres versiones de PSO: una para funciones sin restricciones, una para funciones con restricciones y otra más para problemas de ingeniería estructural. Cabe mencionar que, aunque este último podría incluirse en el apartado de funciones con restricciones, se decidió estudiarlo por separado debido a la complejidad inherente a este tipo de problemas.

7.2. Análisis de parámetros de PSO

Para el análisis de los parámetros de las diferentes versiones de PSO se determinaron cuáles son los parámetros relevantes del algoritmo y sus correspondientes rangos de aplicación (ver tabla 7.1) tomando como referencia los valores propuestos en [84]. Para ello se trabajó con los siguientes algoritmos: CPSO-shake (optimización con restricciones) y Bi-PSO (optimización sin restricciones).

Se realizó un diseño de hipercubos latinos para generar una muestra con cobertura de espacio (*space-filling*) de distintos valores de parámetros para evaluar cada una de las versiones de PSO. Para esta tarea se decidió emplear 20 puntos de diseño que resultaron en 20 configuraciones distintas de parámetros. Sin bien no existe en la literatura una convención sobre la cantidad de puntos a incluir en la muestra, observando los rangos de cada variable (tabla 7.1) y los valores más usados para cada parámetro[**], se concluyó que 20 sería un número adecuado. El diseño LHD se realizó empleando la herramienta estadística

[**]Los parámetros Población y Vecindario son valores discretos con lo cual sólo 10 valores distintos son posibles. Para $\mathcal{C}_i$ y $\mathcal{X}$ se pueden considerar 20 valores distintos ya que en general se utiliza $\mathcal{C}_i$ variando en 0.05 por lo que se tomó como referencia para $\mathcal{X}$. Las probabilidades de mutación son las que pueden tomar mayor cantidad de valores (continuos) por no presentarse ninguna restricción en los valores posibles.

Parámetro	**Rango**
Población	[5, 15]
ProbMutMin	[0, 1]
ProbMutMax	[0, 1]
$\mathcal{C}_i$	[1.0, 1.99]
$\mathcal{X}$	[0.0, 0.99]
Vecindario	[1, 10]
ProbActGauss	[0.0, 0.1]

Tabla 7.1: Rangos de los parámetros de PSO.

R versión 2.5.1[***] y el resultado del muestreo se detalla en la tabla 7.2.

Cada configuración se utilizó para correr independientemente 30 veces cada versión de los algoritmos seleccionados con cada función de prueba. Es decir, Bi-PSO para 13 funciones sin restricciones (optimización global), CPSO-shake para 19 funciones con restricciones (optimización de funciones generales) y CPSO-shake para 3 funciones de ingeniería estructural. Las descripciones de las funciones y sus óptimos correspondientes pueden encontrarse en los Apéndices A, B y C, respectivamente. Los resultados de cada configuración para cada una de las versiones puede encontrarse en el Apéndice G, en donde se especifica: mejor valor encontrado en el total de ejecuciones, valor medio y desviación estándar sobre las 30 ejecuciones. A continuación se describe el análisis para cada caso y se complementa el estudio con una técnica de evaluación gráfica de DOE, las Cajas de Tukey.

7.2.1. Parámetros para funciones sin restricciones

Para f1 cualquier configuración es buena excepto C5 y C8 por los elevados valores medios que se obtuvieron. Tampoco sería conveniente C6, C10, C17 ni C20 ya que aunque las medias no son muy lejanas al óptimo, existen otras configuraciones que resultan más adecuadas porque el mejor valor encontrado coincide con el óptimo y porque tanto la media como la desviación estándar son cero.
Para f2 sucede algo similar que en el caso anterior, todas las configuraciones son adecuadas excepto la C5, C6, C8, C17 y C20.
Para f3 los mejores valores se alcanzaron con las configuraciones C2, C3, C6, C18 y C19. Para C2, el valor medio es elevado al igual que C18, y peor es el caso de C6. Por ello se descartan dichas configuraciones y se eligen C3 y C19. Aunque los valores para C19 son levemente mejores que C3, no se descarta que la última pueda ser adecuada también.
Para f4 el valor más cercano a 0 (el óptimo) se obtuvo con la configuración C19. Con la C11 y C18 los mejores valores son levemente peores que C19 pero sus medias y desviaciones no se alejan tanto de los valores ideales.
Para f5 se puede observar una gran variabilidad de resultados, con medias bastantes alejadas del óptimo y desviaciones pronunciadas. Aunque algunas configuraciones lograron

[***] 2007 The R Foundation for Statistical Computing.

Configuración	Población	ProbMutMin	ProbMutMax	$\mathcal{C}_i$	$\mathcal{X}$	Vecindario	ProbActGauss
C1	11	0.05	0.44	1.49	0.40	10	0.020
C2	9	0.28	0.50	1.01	0.47	6	0.060
C3	13	0.02	0.80	1.30	0.56	7	0.048
C4	12	0.45	0.81	1.05	0.99	8	0.092
C5	6	0.30	0.51	1.83	0.91	3	0.052
C6	7	0.65	0.67	1.92	0.89	4	0.081
C7	14	0.09	0.83	1.23	0.68	6	0.020
C8	9	0.29	0.93	1.51	0.38	5	0.015
C9	11	0.32	0.63	1.37	0.62	5	0.070
C10	5	0.22	0.53	1.43	0.06	1	0.028
C11	10	0.00	0.00	1.70	0.70	3	0.074
C12	12	0.68	0.73	1.88	0.26	2	0.009
C13	8	0.40	0.59	1.27	0.53	7	0.015
C14	13	0.50	0.79	1.56	0.79	9	0.071
C15	8	0.86	0.99	1.77	0.24	2	0.086
C16	15	0.00	0.36	1.96	0.18	9	0.059
C17	7	0.20	0.91	1.70	0.45	3	0.096
C18	14	0.71	0.96	1.11	0.03	9	0.040
C19	10	0.10	0.40	1.80	0.80	3	0.075
C20	6	0.10	0.18	1.74	0.13	4	0.000

Tabla 7.2: Configuraciones para LHD.

encontrar el óptimo (C17) lo hicieron con una desviación alta lo que indica la poca robustez del algoritmo frente a ese conjunto de parámetros. Por tal motivo se eligieron aquellas configuraciones cuyos valores medios y desviaciones son menores: C9, C16, C18 y C19.
Para f6 la mejor configuración es C19 que obtuvo el óptimo con media y desviación cero (encontró el óptimo en las 30 ejecuciones). También podrían ser adecuadas la C2 y C14 porque sus medias son bastante cercanas al óptimo.
Para f7 todas las configuraciones encuentran el óptimo aunque se podrían eliminar aquellas con medias distintas al óptimo (con desviaciones diferentes a cero): C5, C6 y C8.
Para f8 las configuraciones C1, C3, C7, C10, C16 y C19 encontraron el mejor valor conocido, aunque para C1, C3, C7 y C10 las desviaciones fueron altas. Sólo para C16 y C19, las desviaciones fueron aceptables con lo cual se eligieron como posibles configuraciones.
Para f9, los mejores valores se obtuvieron con C19 y C20, siendo la primera la única configuración que alcanzó el óptimo y la de menor desviación.
Para f10, las configuraciones C4, C14, C16 y C19 son las que consiguieron encontrar los óptimos con desviación cero.
Para f11 casi todas las configuraciones encontraron los valores óptimos con lo cual se eligieron aquellas con las que el algoritmo fue más robusto es decir, con las que se obtuvieron las menores desviaciones: C4, C12, C14 y C19.
Para f12 se eligieron las configuraciones con valores medios iguales al óptimo y desviaciones cero: C1, C2, C3, C7, C16, C18 y C19.
Para f13 todas las configuraciones encontraron los óptimos pero se eligen sólo aquellas con medias más cercanas al mejor valor conocido y desviación mínima (casi cero): C1 y

Función	Configuración
f1	todas-{C5,C6,C8,C10,C17,C20}
f2	todas-{C5,C6,C8,C17,C20}
f3	C3,C19
f4	C11,C18,C19
f5	C9,C16,C18,C19
f6	c2,C14,C19
f7	todas-{C5,C6,C8}
f8	C16,C19
f9	C19,C20
f10	C4,C14,C16,C19
f11	C4,C12,C14,C19
f12	C1,C2,C3,C7,C16,C18,C19
f13	C1,C19

Tabla 7.3: Mejores configuraciones para las funciones sin restricciones.

C19.
Un resumen de las configuraciones para cada función se muestra en la tabla 7.3.

Como puede observarse en la tabla 7.3, la configuración 19 es la que se repite en todas las funciones como una de las mejores opciones de parámetros. La C16 es otra configuración que se detecta en muchas funciones al igual que la C18. Para decidir qué configuración utilizar en este tipo de funciones sin restricciones, se podría observar la tabla y decidir que la C19 es una de las mejores, pero para asegurar esta decisión se plantea un nuevo análisis a continuación.
Se realizaron cajas de Tukey para la mayor parte de funciones en las que varias configuraciones son candidatas a ser las mejores. Para otras funciones como f1, f2 y f7 no se realizó este análisis ya que casi cualquier configuración obtuvo buenos resultados. Para otras funciones como f10 y f12 los valores obtenidos fueron los mejores conocidos con desviación cero, o sea cualquier configuración es buena, con lo cual ninguna caja de Tukey se requiere para seleccionar la configuración adecuada.

La figura 7.2 muestra las cajas de Tukey para las dos configuraciones propuestas para f3. Puede observarse que la mediana (línea más gruesa en el gráfico) de C19 es más baja que la de C3, el primer y tercer cuartil son equilibrados y el ancho de la caja es mínimo lo que indica que la mayoría de los datos de la muestra se encuentran muy cercanos a la mediana (sinónimo de robustez del algoritmo, al tenerse menor dispersión de los valores obtenidos). No sucede lo mismo con la caja de C3 que posee cuartiles desequilibrados indicando una mayor cantidad de datos en el tercer cuartil con valores extremos alejados (tope del bigote). Para ambas configuraciones se detectó la presencia de outliers. Visualizando los datos provistos por el gráfico se concluye que C19 es una buena configuración para f3.

Observando la figura 7.3 correspondiente a las cajas de Tukey para f4, podría descartarse la C18 por la presencia de outliers, pero la selección de C11 por sobre C19 o viceversa no es clara. Tanto C11 como C19 tienen dispersiones de datos similares, sesgos pronunciados observados en las medianas descentradas y valores extremos en los bigotes

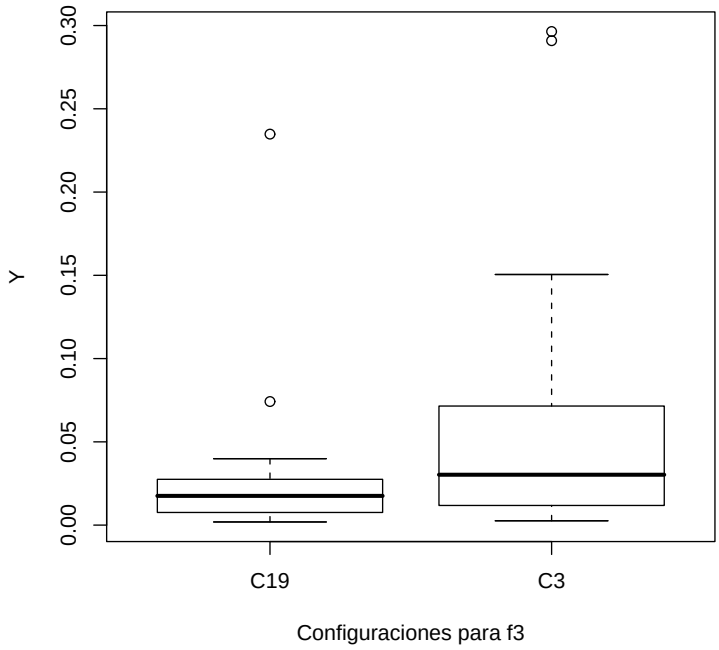


Figura 7.2: Cajas de Tukey de las configuraciones para f3.

largos. Tanto C11 como C19 pueden ser utilizadas como configuraciones posibles.

Para la función f5 la figura 7.4 muestra grandes dispersiones y sesgos para todas las configuraciones posibles. C16 y C18 casi no poseen bigotes, pero las cajas son un poco más amplias que para C19 y C9. C19 posee sólo dos outliers y las demás configuraciones al menos 4. Basando la elección sólo en ese hecho es que se selecciona C19 como posible configuración para f4.

Las cajas de Tukey de f6 mostradas en la figura 7.5 muestran que la dispersión de los valores de la muestra con respecto a la mediana es nula lo cual indica que el algoritmo se ha comportado robustamente con esas configuraciones. Se descarta la C2 por poseer un outlier. Se concluye que tanto C14 como C19 son buenas configuraciones para f6.

Un caso similar al anterior sucede con f8 cuyas cajas de Tukey se muestran en la figura 7.6. Sólo por tener valores outliers se descarta C16 y se concluye que C19 es la mejor configuración.

Observando la figura 7.7 correspondiente a las cajas para f9, la C19 no posee tanta dispersión como C20 y los valores extremos son inferiores que los de C20. Por tal motivo, se selecciona C19 como la mejor configuración para f9.

Para f11 las cajas para C12 y C19 en la figura 7.8 son muy similares aunque para C12 se detecta un outlier y el bigote superior es mayor que el de C19. Las configuraciones C14 y C4 no son muy elegibles debido a que sus medianas se encuentran en la parte inferior de las cajas indicando la mala dispersión de los datos. Se detectaron dos outliers en estas dos últimas configuraciones. Por estos motivos se descartan y se concluye con la selección

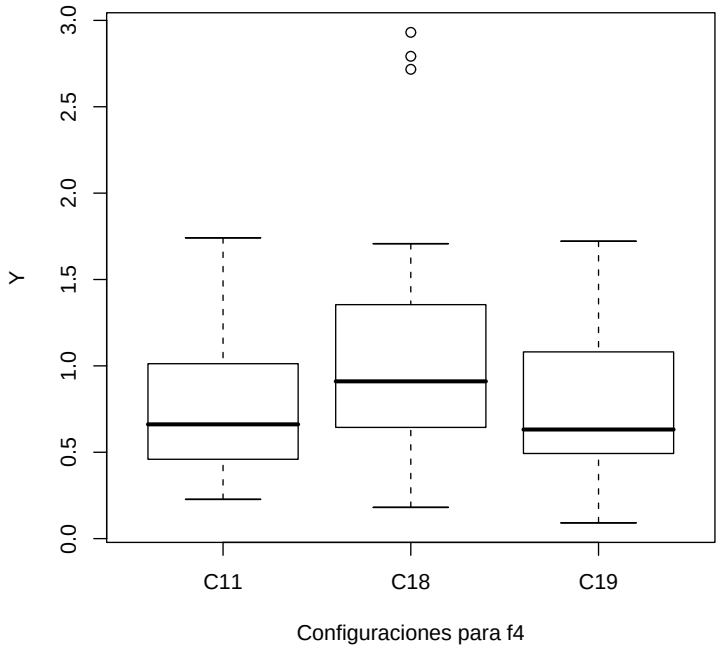


Figura 7.3: Cajas de Tukey de las configuraciones para f4.

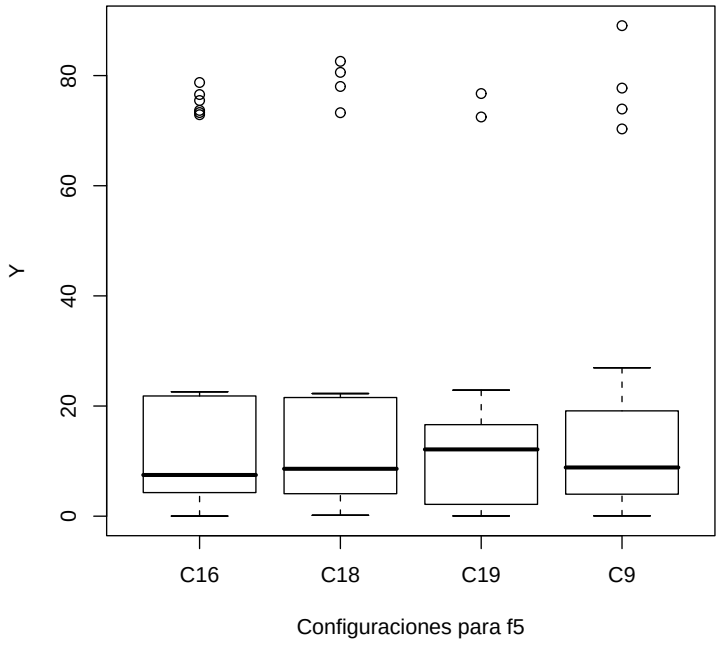


Figura 7.4: Cajas de Tukey de las configuraciones para f5.

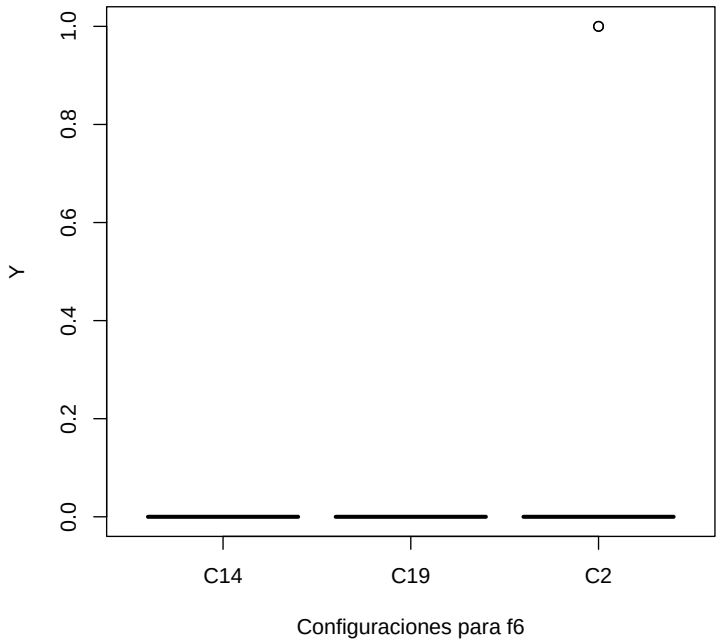


Figura 7.5: Cajas de Tukey de las configuraciones para f6.

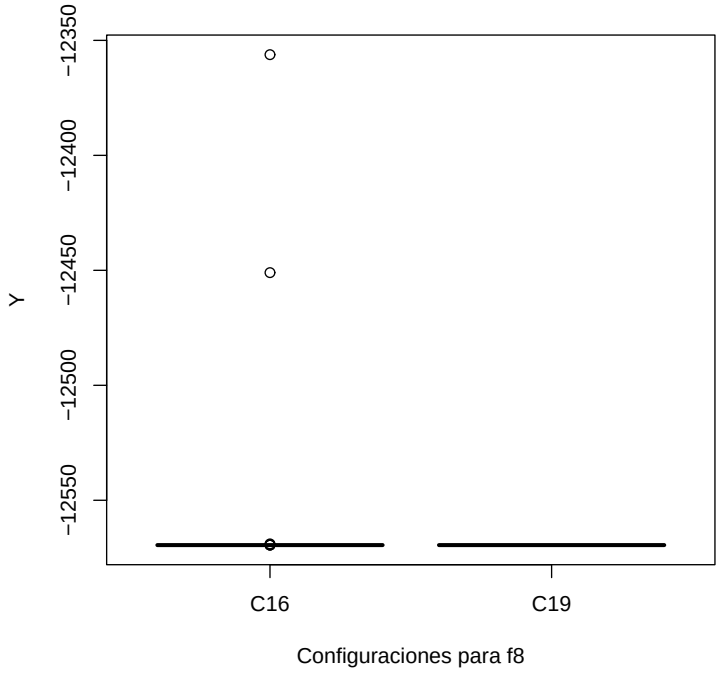


Figura 7.6: Cajas de Tukey de las configuraciones para f8.

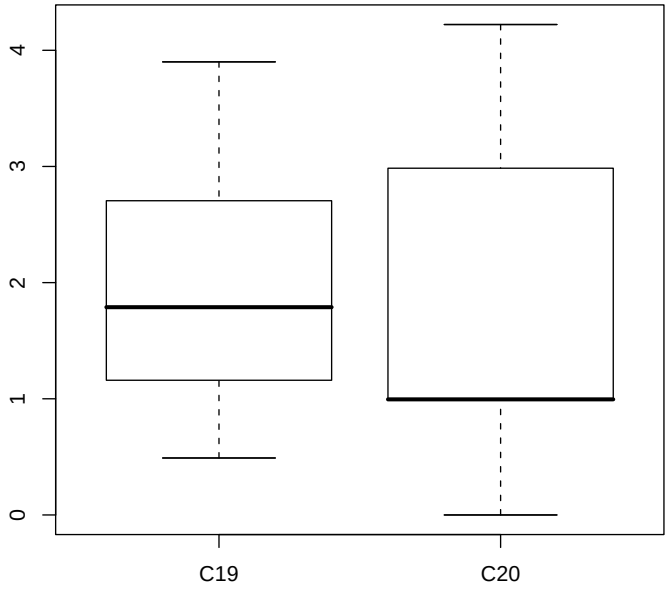


Figura 7.7: Cajas de Tukey de las configuraciones para f9.

de C19.

Para f13, las cajas de Tukey de la figura 7.9 indican que tanto C1 como C19 son elegibles como configuraciones posibles. La dispersión de los valores con respecto al valor de la mediana es nula, no tienen sesgo ni bigotes y sí poseen ambas un valor outlier. Es indistinto por lo tanto elegir C1 o C19.

Como conclusión del análisis gráfico de las distribuciones de los valores encontrados con las distintas configuraciones de parámetros, se opta por fijar los parámetros en C19. Tanto a través del análisis directo de los resultados visibles en las tablas (Apéndice G) como el estadístico observado en los gráficos, es posible argumentar que para funciones generales sin restricciones la configuración 19 del algoritmo Bi-PSO arrojará buenos resultados. De esta forma se concluye en la fijación de los parámetros en los valores de C19 que se resumen en la tabla 7.4.

Población	**ProbMutMin**	**ProbMutMax**	$\mathcal{C}_i$	$\mathcal{X}$	**Vecindario**	**ProbActGauss**
10	0.10	0.40	1.80	0.80	3	0.075

Tabla 7.4: Configuración de PSO para funciones sin restricciones.

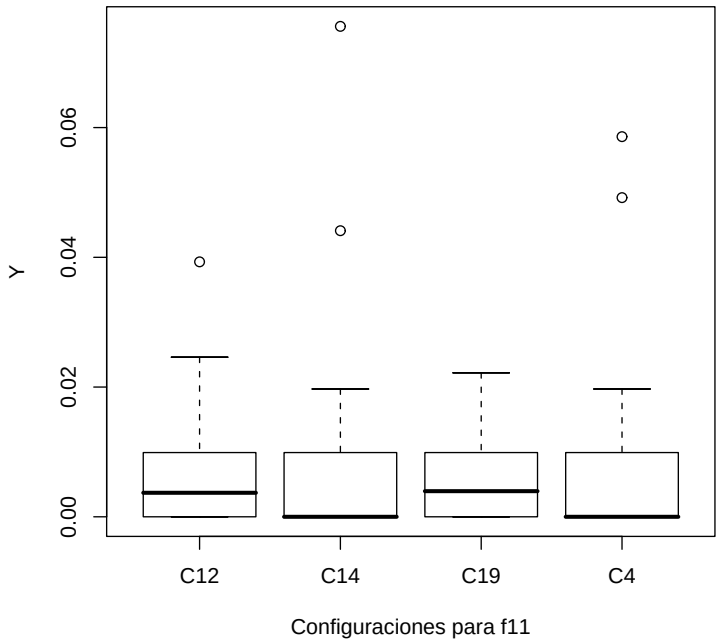


Figura 7.8: Cajas de Tukey de las configuraciones para f11.

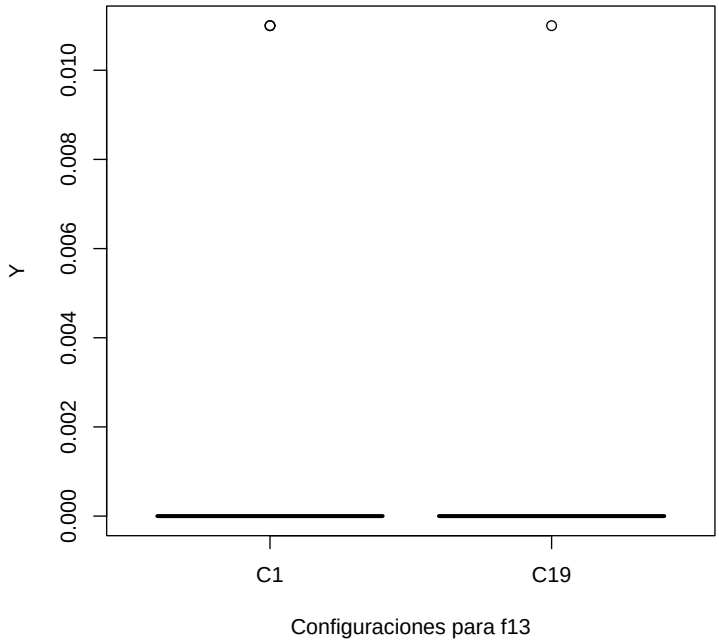


Figura 7.9: Cajas De Tukey de las configuraciones para f13.

7.2.2. Parámetros para funciones con restricciones

De los resultados arrojados con las diferentes configuraciones (ver Apéndice G) se puede apreciar que para g1 la mayor parte de configuraciones provoca que el algoritmo encuentre el óptimo, excepto para C1, C3, C9 y C14 con las que, aunque los mejores encontrados no son los óptimos, son bastante cercanos (desviaciones pequeñas).
Casi todas las configuraciones evaluadas con la función g2 alcanzaron el mejor valor conocido y en sólo tres, los valores son lo suficientemente cercanos al óptimo como para no incluirlos como posibles configuraciones. Por ello, para esta función no se puede concluir la utilización de una configuración de parámetros en particular.
Para g3 sucede lo mismo que para g2, todas las configuraciones arrojaron buenos valores óptimos con lo cual puede concluirse que ninguna combinación de parámetros es mejor a otra.
Las soluciones encontradas con los diferentes conjuntos de parámetros para g4 no resultaron factibles en la mitad de ellas. De las soluciones factibles sólo tres (C3, C6 y C19) son las que más se aproximan al óptimo. C3 posee la mayor desviación de las tres configuraciones posibles, C6 un poco menor y C19 la menor de todas por lo que se selecciona esta última como la más adecuada.
En general, para g5 las desviaciones estándar son altas para todas las configuraciones, aunque para algunos casos los mejores valores encontrados se encuentran próximos al óptimo. Este es el caso de C2, C9, C15, C16, C18 y C19. Observando los valores medios y las desviaciones se puede evidenciar la dificultad del algoritmo en cuanto a robustez ya que encuentra valores cercanos al óptimo pero con grandes desviaciones.
Para g6 los mejores valores se aproximan al óptimo pero no alcanzan a acercarse demasiado. Sólo C19 obtiene valores óptimos cercanos al benchmark, aunque con desviación elevada.
Con la función g7 sucede algo similar a la función anterior, sólo una configuración obtiene un valor cercano al mejor conocido, con media próxima al mejor valor y desviación aceptable, la C19.
Todas las configuraciones probadas en g8 obtuvieron el óptimo con desviación cero o casi cero, por ello cualquier combinación de parámetros es aceptable como configuración adecuada.
Aunque todos los valores mejores se acercan bastante al óptimo, para g9 se seleccionaron aquellas configuraciones con mejores valores más cercanos al óptimo y menores desviaciones, es decir, C11, C12, C15, C16, C18 y C19.
Las desviaciones obtenidas para g10 demuestran la dificultad del algoritmo en obtener soluciones cercanas al óptimo en todas las ejecuciones. Los valores medios en general están alejados del mejor encontrado. Se eligen las configuraciones que lograron encontrar valores similares al óptimo como son C8, C11 y C19.
Para g11 cualquier configuración obtiene el valor óptimo con desviación cero o casi cero, lo cual presume que la selección de una configuración determinada no tiene influencia en el desempeño del algoritmo.
Lo mismo que para el caso de la función anterior, para g12 cualquier configuración encuentra el benchmark con desviación cero.
La función g13 es una que se caracteriza por su dificultad, y ello se ve reflejado en los

mejores valores obtenidos que difieren bastante del óptimo, excepto para C10, C16 y C19.
Para g14, los valores obtenidos varían entre las diferentes configuraciones y los valores medios son relativamente distintos de los mejores lo que presupone la dificultad del algoritmo en encontrar buenos valores en el total de ejecuciones. Las dos configuraciones que obtuvieron valores cercanos al benchmark son C3 y C19.
Más de la mitad de configuraciones no lograron obtener soluciones factibles para g15. De las factibles se seleccionaron aquellas que obtuvieron valores más próximos al benchmark y con mínima desviación: C3, C5, C6, C9 y C19.
Para g16 tres configuraciones encontraron el óptimo conocido con desviación cero, con lo cual se seleccionaron como las más adecuadas: C12, C16 y C19.
El comportamiento de las diversas configuraciones para g17 evidencia que es posible encontrar buenas soluciones pero con desviaciones elevadas. Si se observa la tabla G.30 los mejores valores son bastante similares (y cercanos al benchmark) y las medias alejadas de los mejores. Las desviaciones varían de 3 a 115 puntos. Con todo lo expuesto resulta difícil seleccionar un conjunto de parámetros como el más adecuado.
Una buena cantidad de configuraciones encuentra el mejor valor conocido o valores muy similares para g18. Se descartan sólo dos conjuntos de parámetros (C3 y C14) por no aproximarse tanto como los otros casos al mejor conocido, pero en general cualquier configuración podría emplearse.
Para g19 los mejores valores encontrados varían entre 37 y 49, no logrando aproximar el óptimo (32.655). Los valores medios oscilan entre 52 y 177 lo que no permite seleccionar una configuración adecuada para esta función. Se concluye luego, que cualquier selección de parámetros podría obtener un valor cercano al óptimo pero con una desviación considerable.
Un resumen de las configuraciones para cada función se muestra en la tabla 7.5.

La configuración 19 es una de las que se repite en la totalidad de funciones como una de las mejores opciones de parámetros, según consta en la tabla 7.5. Otras que se repiten son la C3, C6 y C16, y en varias funciones como g1, g2, g8, g11, g12, g17, g18 y g19 casi es imposible determinar qué configuración es la adecuada ya que todas obtuvieron resultados aceptables. Basados en estos resultados, se determinará a través de la utilización de las cajas de Tukey cuál sería una configuración apta, estudiando las características de la distribución de los datos en aquellas funciones en las que varios conjuntos de parámetros obtienen buenos resultados. Se decide no realizar las cajas de Tukey de las funciones anteriormente nombradas ya que cualquier configuración es buena, sólo se realizarán las cajas para las demás funciones y con base en esos resultados se fijará la configuración que se considere más adecuada.
Para g3 las cajas de la figura 7.10 indican que C19 posee la mediana más baja y el sesgo es simétrico (mediana centrada en la caja) aunque los bigotes son largos, indicando que las colas de la distribución son amplias. Las cajas de C3 y C9 muestran amplia dispersión de datos (alturas pronunciadas) y C1 posee sesgo desigual con valores outliers. La configuración C19 parece ser la más promisoria.

Las cajas de Tukey de la figura 7.11 para g4 ilustran que no hay dispersión de datos ya que las medianas se observan en cajas de altura cero. C6 posee unos pequeños bigotes

Función	Configuración
g1	todas-{C1,C3,C9,C14}
g2	todas
g3	C1,C3,C9,C19
g4	C3,C6,C19
g5	C2,C9,C15,C16,C18,C19
g6	C19
g7	C19
g8	todas
g9	C11,C12,C15,c16,c18,c19
g10	C8,C11,C19
g11	todas
g12	todas
g13	C10,c16,C19
g14	C3,C19
g15	C3,C5,C6,C9,C19
g16	C12,C16,C19
g17	todas
g18	todas-{C3,C14}
g19	todas

Tabla 7.5: Mejores configuraciones para las funciones con restricciones.

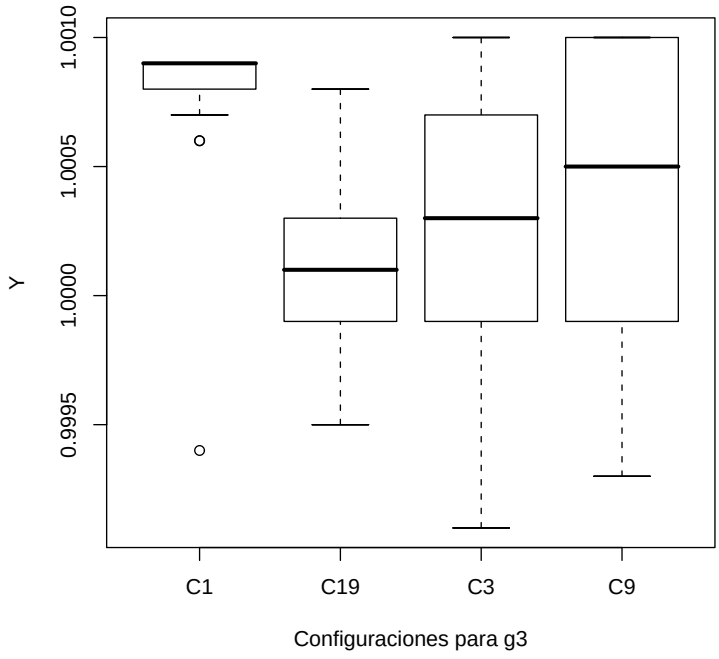


Figura 7.10: Cajas de Tukey de las configuraciones para g3.

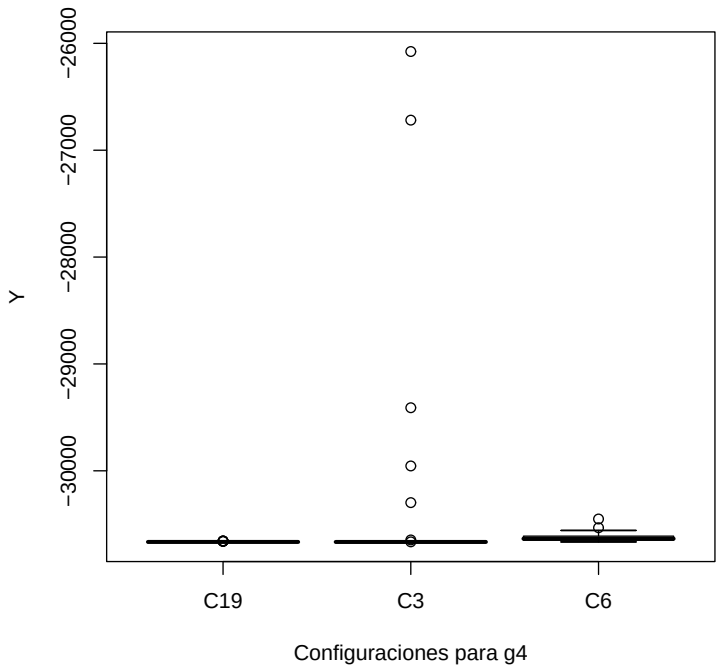


Figura 7.11: Cajas de Tukey de las configuraciones para g4.

evidenciando una pequeña cola en la distribución. C19 posee un valor outlier, C3 posee al menos 6 y C6 dos valores alejados de los demás de la muestra. Nuevamente, C19 parece ser la mejor configuración.

En la figura 7.12 las cajas para g5 muestran una gran dispersión de datos con largas colas en la parte superior. Debido a la dificultad del algoritmo para encontrar buenas soluciones y de una manera robusta, la selección de una configuración adecuada resulta difícil. La C15 es la que presenta mejor dispersión y sesgo, aunque con la presencia de outliers. C19 posee el bigote superior más corto indicando la corta cola en la distribución de valores y no se observan outliers, aunque posee un sesgo importante. Debido a las primeras características observadas, se selecciona C19 junto a C15 como posibles configuraciones.

Como puede observarse en la figura 7.13, la configuración C19 presenta buenas características comparada a las otras para g9. La caja muestra baja dispersión de datos con un sesgo equilibrado y sin outliers, por ello se la selecciona como la más adecuada para g9.

Las cajas de Tukey para g10 se muestran en la figura 7.14. La configuración C19 presenta una buena dispersión y adecuado sesgo de valores. Los bigotes no son más grandes que los observados en las otras configuraciones y no existen outliers para este conjunto de parámetros con lo que se concluye en la selección de C19 para esta función.

La figura 7.15 detalla el estudio muestral para g13. Aunque C16 y C19 parecerían ser similares en cuanto a sesgo y dispersión de los datos, C19 no presenta el bigote inferior pero C16 posee un bigote más pequeño en la parte superior. C10 muestra mayor dispersión de datos y colas de distribución más pronunciadas. Se concluye que tanto C16 como C19

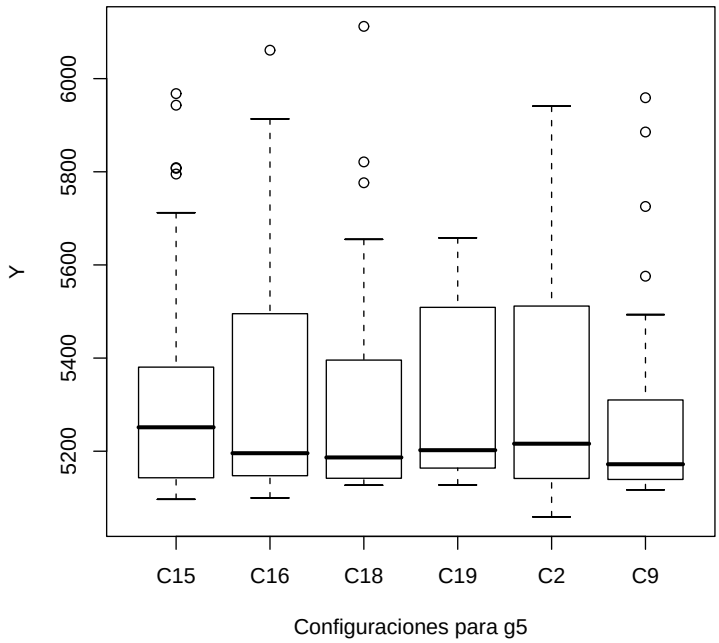


Figura 7.12: Cajas de Tukey de las configuraciones para g5.

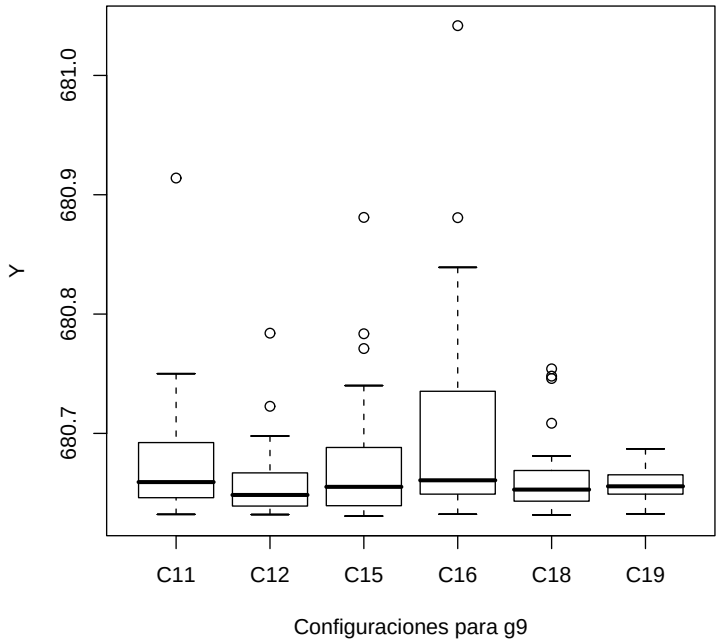


Figura 7.13: Cajas de Tukey de las configuraciones para g9.

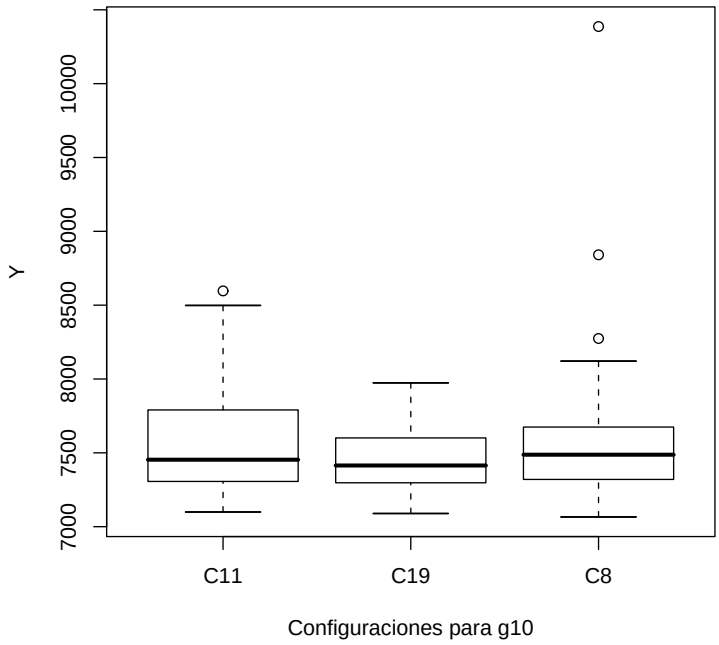


Figura 7.14: Cajas de Tukey de las configuraciones para g10.

podrían ser configuraciones posibles para g13.

Las cajas para g14 en la figura 7.16 son similares en cuanto a sesgo y distribución de datos, aunque la mediana de C3 es levemente superior y el bigote inferior muestra una de las colas más pronunciadas. C3 también presenta un valor outlier. Se elige, luego, C19 como la configuración más adecuada.

La figura 7.17 ilustra que C19 debería ser elegida para g15 como una configuración adecuada debido a que no presenta sesgo, ni dispersión de datos lo cual presupone la robustez del algoritmo en encontrar el óptimo en el total de ejecuciones. Algunos outliers se muestran para esta configuración, al igual que para las demás. C9 podría ser seleccionada también aunque se observa un poco de dispersión de datos.

Con similares características que para g15, la caja para g16 no muestra sesgo ni dispersión, sólo un valor outlier bastante cercano a la mediana. La configuración C12 también podría ser seleccionada, aunque su valor outlier está levemente alejado de la mediana. La figura 7.18 ilustra las cajas de Tukey para estas configuraciones.

Como conclusión general del análisis gráfico de las distribuciones de los valores encontrados con las diferentes configuraciones propuestas, se opta por fijar los parámetros en C19. Tanto a través del análisis directo de los resultados (Apéndice G) como del análisis estadístico observado en las cajas de Tukey, es posible argumentar que para funciones generales con restricciones la configuración C19 del algoritmo CPSO-shake arrojará buenos resultados y de esa forma se concluye en la fijación de estos parámetros. En la tabla 7.6, se resumen dichos valores para cada parámetro del algoritmo.

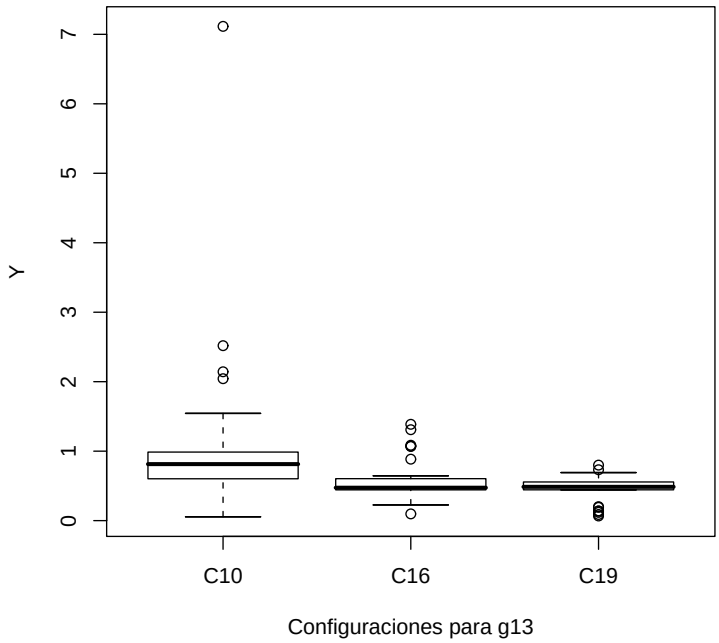


Figura 7.15: Cajas de Tukey de las configuraciones para g13.

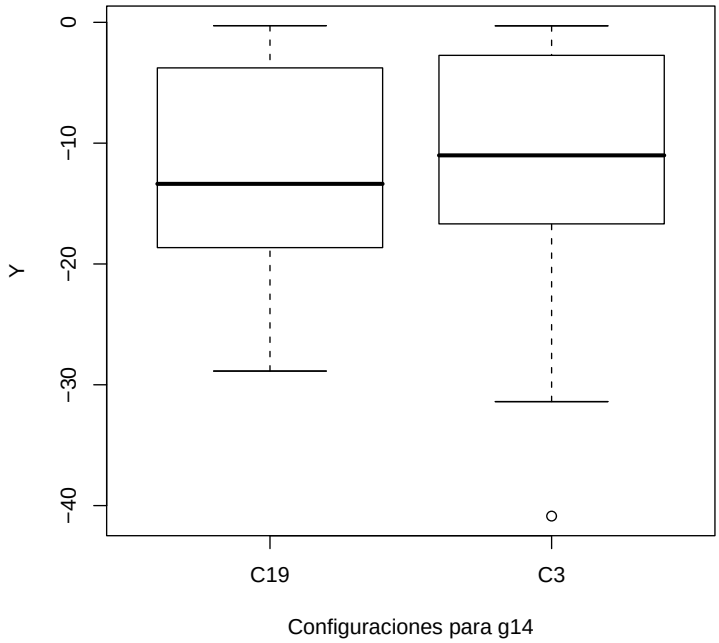


Figura 7.16: Cajas de Tukey de las configuraciones para g14.

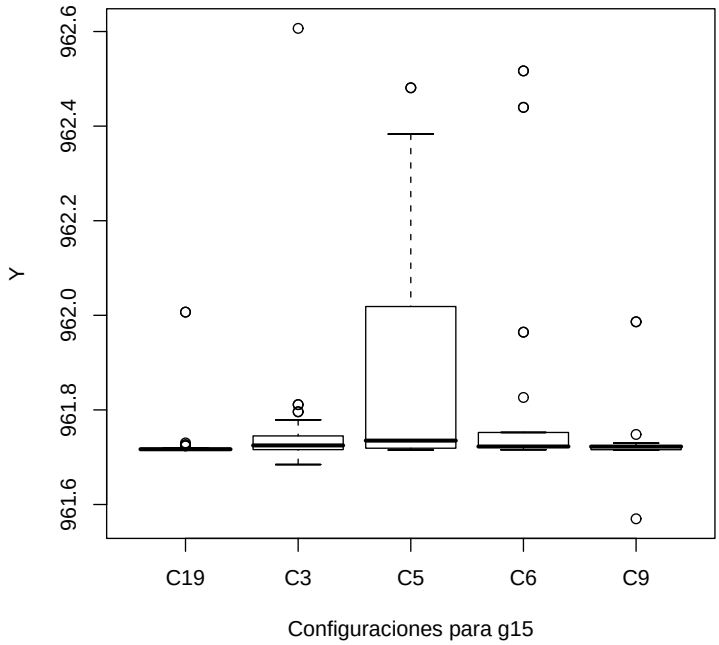


Figura 7.17: Cajas de Tukey de las configuraciones para g15.

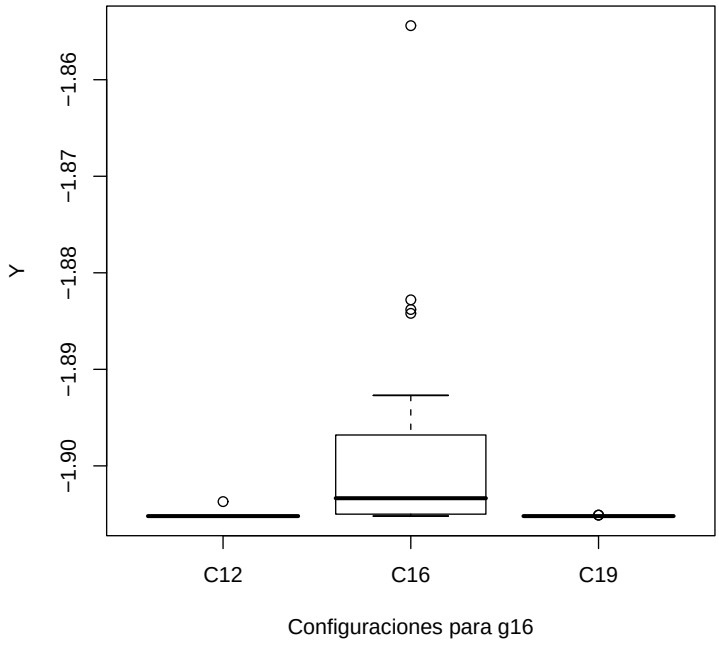


Figura 7.18: Cajas de Tukey de las configuraciones para g16.

Población	ProbMutMin	ProbMutMax	$\mathcal{C}_i$	$\mathcal{X}$	Vecindario	ProbActGauss
10	0.10	0.40	1.80	0.80	3	0.075

Tabla 7.6: Configuración de PSO para las funciones con restricciones.

7.2.3. Parámetros para las funciones de ingeniería estructural

Las funciones de ingeniería estructural podrían ser encuadradas como funciones con restricciones pero al ser difíciles de optimizar, posiblemente por ser problemas ingenieriles reales, se prefirió realizar un estudio de parámetros particular para este caso. Esta decisión fue tomada luego de probar el conjunto de parámetros fijado en la tabla 7.4 y no habiendo encontrado los resultados esperados. Las tres funciones: armadura plana de 10 barras, armadura espacial de 25 barras y armadura plana de 200 barras son referenciadas a continuación como t1, t2 y t3 respectivamente.
Observando el Apéndice G, los mejores valores para la función t1 fueron encontrados con C11 y C16 con desviaciones por encima de 3 puntos. Otras configuraciones se aproximan al óptimo pero con valores medios mayores a estas dos configuraciones.
Para t2 los mejores valores se obtuvieron con las configuraciones C10, C11 y C17 aunque éstas no poseen las mejores medias. Para estudiar la robustez del algoritmo se agrega C2 y C8 ya que, aunque con ellas se encontraron valores un poco mayores a los de las otras configuraciones, poseen medias y desviaciones más pequeñas que el resto de configuraciones.
Las configuraciones que encontraron mejores valores para t3 fueron C4 y C11. Las demás se alejan bastante de esos mejores. Aunque C4 presenta una desviación superior a C11, se incluye igualmente como posible configuración para esta función.
El resumen de configuraciones para cada función de ingeniería es mostrado en la figura 7.7.

Función	Configuración
t1	C11,C16
t2	C2,C8,C10,C11,C17
t3	C4,C11

Tabla 7.7: Mejores configuraciones para funciones de ingeniería estructural.

Como puede observarse en la tabla 7.7 la configuración que se repite en las tres funciones es la C11, con lo cual se podrían fijar los parámetros en los valores establecidos por ese conjunto, pero para completar el estudio, se presenta a continuación el estudio estadístico de las distribuciones de cada una de estas configuraciones.
Observando la figura 7.19 se puede localizar la mediana de C11 casi en el centro de la caja, lo cual indica un sesgo equilibrado. La dispersión de C11 es levemente inferior que la que presenta C16. Los bigotes de C11 son bastante más cortos que los de C16 lo cual indica que C11 posee colas en la distribución mucho más pequeñas. Con esto se concluye que C11 es la configuración adecuada para t1.

La figura 7.20 muestra las cajas de Tukey para t2. De las tres configuraciones que obtuvieron los mejores valores C10, C11 y C17, sólo C11 es la que no tiene dispersión

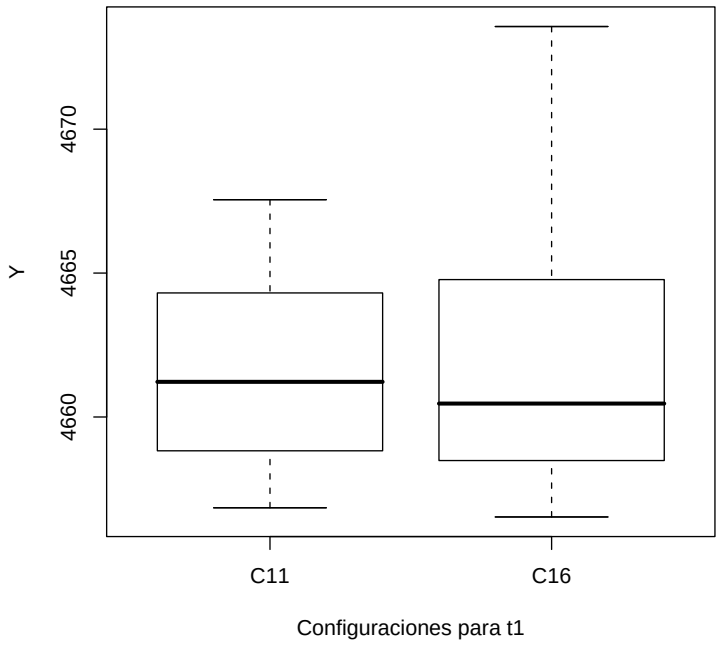


Figura 7.19: Cajas de Tukey de las configuraciones para t1.

de datos, aunque C17 posee una muy pequeña. Las otras dos configuraciones que se estudiaron para detectar qué distribución poseen sus datos fueron C2 y C8. Como se ilustra en el gráfico, las cajas son similares a la de C11 (sin altura) lo que muestra robustez en el algoritmo al encontrar las soluciones. Algunos valores outliers se detectan para las configuraciones (excepto C10). Como conclusión, las configuraciones C11, C2 y C8 son adecuadas para t2.

Las cajas de Tukey para t3 se presentan en la figura 7.21. Como puede observarse, la caja de C11 posee un sesgo un poco menor que C10 y C17, y la distribución de los datos es más pareja. Los bigotes inferiores de C10 y C11 son similares. No así los de C17 evidenciando colas en la distribución más pronunciadas. Tanto C10 como C17 poseen un valor outlier. Como conclusión para esta función, tanto C10 como C11 podrían ser seleccionadas como configuraciones adecuadas ya que los mejores valores obtenidos varían muy poco (se alejan más de C17) y las cajas tienen bastante similitud.

Como conclusión del estudio gráfico de las distribuciones de datos para las funciones de ingeniería estructural, se propone la configuración C11 como la más adecuada para este tipo de funciones. En la tabla 7.8 se resumen los valores para cada parámetro del algoritmo.

Si se observan detenidamente las tablas 7.6 y 7.8, se puede ver la similitud de valores de los parámetros en el tamaño de población y el vecindario, siendo ligeramente diferentes los factores de aprendizaje, factor de constricción y probabilidad de mutación Gaussiana. La probabilidad de mutación dinámica (mínima y máxima) no está presente en esta versión

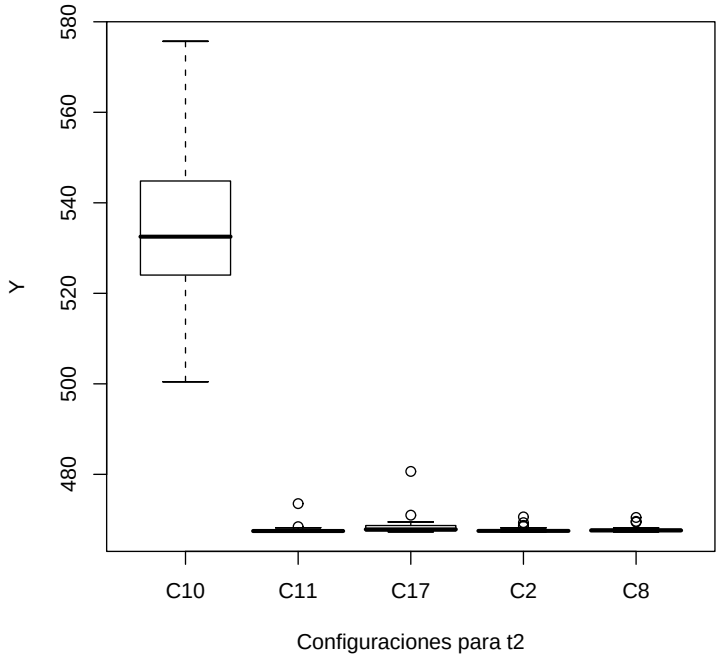


Figura 7.20: Cajas de Tukey de las configuraciones para t2.

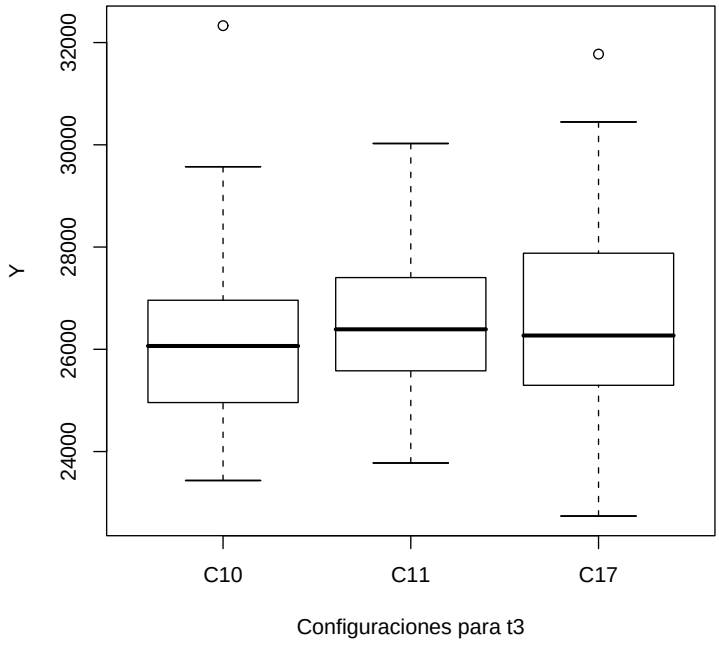


Figura 7.21: Cajas de Tukey de las configuraciones para t3.

Población	**ProbMutMin**	**ProbMutMax**	$\mathcal{C}_i$	$\mathcal{X}$	**Vecindario**	**ProbActGauss**
10	0.00	0.00	1.70	0.70	3	0.074

Tabla 7.8: Configuración de PSO para las funciones de ingeniería estructural.

de PSO. Esto posiblemente se deba a que las funciones de ingeniería tienen restricciones, por lo tanto comparten con estas últimas muchas características y así como en el apartado anterior se detectó que la mejor configuración era la de la tabla 7.6, en este caso se repiten prácticamente los valores antes encontrados. La diferencia más pronunciada entre ambos casos es que para las funciones generales con restricciones fue necesaria la aplicación de un operador de mutación mientras que en el caso de las funciones ingenieriles no. Tal vez la dificultad de este tipo de problemas y las acotadas evaluaciones que se requieren habitualmente para su resolución provoquen que si se aplicara mutación dinámica, la población se podría diversificar demasiado evitando la convergencia en un tiempo considerablemente corto. Este es un aspecto importante que se debe tener en cuenta, basándose en las características de las funciones a optimizar, con lo cual no siempre es posible generalizar la utilización de un conjunto de parámetros fijos. Obviamente éste es uno de los mayores inconvenientes que se presentan en los problemas de optimización.
Como conclusión se propone el empleo de ciertos conjuntos de parámetros para funciones con y sin restricciones, pero si algún otro tipo de funciones se intentaran resolver usando dichos parámetros y no se encontraran resultados adecuados, sería necesario plantear un nuevo estudio de ajuste de parámetros.

Capítulo 8

Conclusiones y Trabajos Futuros

En este capítulo se describen las conclusiones generales que se derivan de todo el trabajo realizado, así como también algunos lineamientos para futuras extensiones.

Contenido del Capítulo

8.1. Conclusiones Generales

El paradigma de Inteligencia Colectiva continúa siendo hoy en día, un área prometedora en lo referente a optimización de todo tipo de problemas. También es un área que se encuentra muy activa en cuanto a investigación, buscando mejorar y extender los algoritmos derivados del paradigma.
La heurística *Particle Swarm Optimization* (PSO) es uno de los algoritmos más representativos del paradigma de Inteligencia Colectiva, y que ha demostrado, desde sus inicios en el año 1995, ser muy competitivo en una amplia gama de problemas de optimización.

En este trabajo se presentaron varias versiones de la heurística PSO para solucionar problemas que poseen diversas características.

La propuesta de PSO para problemas mono-objetivo sin restricciones (Bi-PSO) se validó con un conjunto de 13 funciones muy utilizadas en la literatura especializada, debido a su alta dimensionalidad y variadas características (en cuanto a multimodalidad y separabilidad). Los resultados de esta propuesta fueron comparados con los obtenidos con un algoritmo basado en Evolución Diferencial, y con otro basado en PSO. Las conclusiones demostraron que Bi-PSO es superior en desempeño al algoritmo basado en PSO, y que es comparable con el algoritmo basado en Evolución Diferencial, a pesar de que algunas desviaciones alteran la robustez de Bi-PSO.

La versión CPSO-shake fue propuesta para resolver problemas mono-objetivo con restricciones. Para validar su desemepeño se utilizaron: un benchmark de 24 funciones, 3 problemas de diseño óptimo de armaduras, 4 problemas de diseño ingenieril de piezas y un problema de cargas eléctricas en sistemas de potencia. Para el caso de las 24 funciones, CPSO-shake obtuvo resultados competitivos. Para las funciones de diseño de armaduras, la propuesta mejoró los óptimos conocidos. Para las funciones de diseño ingenieril de piezas, CPSO-shake obtuvo las mejores soluciones conocidas pero con una menor cantidad de evaluaciones (menor esfuerzo computacional). Para estos tres últimos casos, si bien los resultados fueron los esperados, la robustez del algoritmo es un aspecto que debe ser estudiado en mayor detalle. Para el problema de despacho de cargas eléctricas, CPSO-shake obtuvo el mejor resultado conocido en los dos casos más simples, mientras que para el caso más complejo, obtuvo un valor mejor que el mejor conocido. El aporte radica, fundamentalmente, en la rápida respuesta de CPSO-shake para resolver este problema, al comparársele con los tiempos requeridos por las otras técnicas que se han aplicado a este caso de estudio.

En este trabajo se propuso la incorporación de un algoritmo PSO en una eficaz técnica matemática (*epsilon-constraint*) resultando en el método híbrido *epsilon-CPSO*, el cual se utilizó para resolver problemas multiobjetivo difíciles. Si bien existe una buena cantidad de algoritmos que resuelven problemas multiobjetivo estándar, existen en la literatura especializada algunos problemas con varios objetivos que son muy difíciles de resolver para la mayoría de los algoritmos evolutivos existentes. epsilon-CPSO se validó con problemas

difíciles de 2 y 3 funciones objetivo. Los resultados obtenidos demostraron que la técnica propuesta pudo obtener frentes de Pareto de alta calidad, en comparación a los obtenidos con técnicas del estado del arte en optimización evolutiva multiobjetivo como es NSGA-II. Los resultados obtenidos también fueron comparados con los de un método híbrido similar al propuesto, pero que utiliza un algoritmo cultural en vez de un PSO. Las conclusiones de esta comparación fueron que la propuesta híbrida con PSO es superior a la híbrida con el cultural, en varios problemas (comparación indirecta).

Los resultados observados en las comparaciones indirectas, así como, el análisis estadístico de las observaciones directas, demuestran que los algoritmos propuestos son adecuados para resolver una cantidad considerable de problemas mono y multiobjetivo, tanto en lo referente a la calidad de los resultados como en lo concerniente al tiempo de respuesta utilizado para obtener los mejores valores.

8.2. Trabajos Futuros

Una de las futuras extensiones de este trabajo es la realización de un estudio detallado de la robustez de los algoritmos propuestos, a fin de detectar los causantes de la variabilidad excesiva que se experimenta en algunos de los resultados obtenidos.

También se pretende realizar otras pruebas estadísticas para averiguar no sólo las configuraciones de parámetros ideales, sino también las posibles interacciones entre dichos parámetros.

Algunas pruebas de epsilon-CPSO con un mayor número de evaluaciones y el mismo conjunto de problemas deberían realizarse, a fin de corroborar si el desempeño del método híbrido se puede mejorar. El número de evaluaciones utilizado en este trabajo fue fijado en las cantidades empleadas para poder realizar una comparación justa con el método híbrido con el algoritmo cultural, pero puede resultar demasiado reducido por tratarse de optimización de problemas multiobjetivo.

Otras técnicas de interpolación podrían utilizarse en el proceso de aumentar la calidad de los frentes obtenidos con el método híbrido, a fin de concluir cuál sería la más adecuada para la mayoría de los problemas resueltos.

Finalmente, la versión multiobjetivo debería ser evaluada con problemas de más de 3 funciones objetivo, a fin de corroborar si su desempeño continúa siendo el adecuado o si se requiere introducir cambios al algoritmo.

Apéndice A

Funciones mono-objetivo sin restricciones con 30 variables de decisión

$f1$ **- Modelo de la Esfera**: función unimodal y separable.

$$f1(x) = \sum_{i=1}^{30} x_i^2$$

$$-100 \leq x_i \leq 100 \qquad\qquad min(f1) = f1(0, \cdots, 0) = 0$$

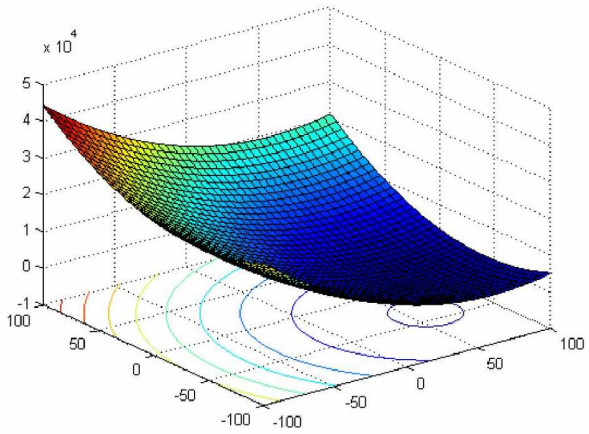


Figura A.1: Función Modelo de la Esfera.

$f2$ **- Schwefel 2.22**: función unimodal y separable.

$$f2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|$$

$$-10 \leq x_i \leq 10 \qquad\qquad min(f2) = f_2(0, \cdots, 0) = 0$$

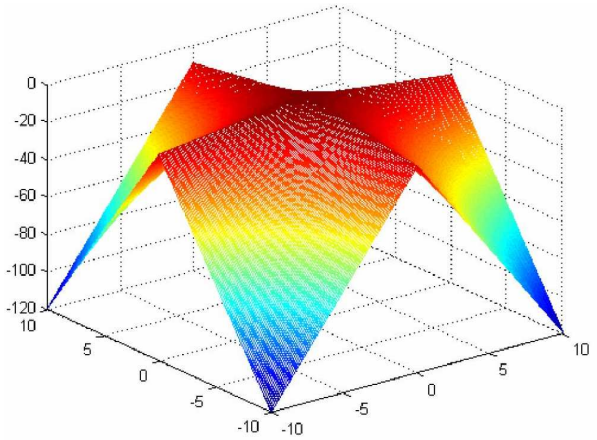


Figura A.2: Función Schwefel 2.22.

$f3$ **- Schwefel 1.2**: función unimodal y no separable.

$$f3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2$$

$$-100 \leq x_i \leq 100 \qquad\qquad min(f3) = f3(0, \cdots, 0) = 0$$

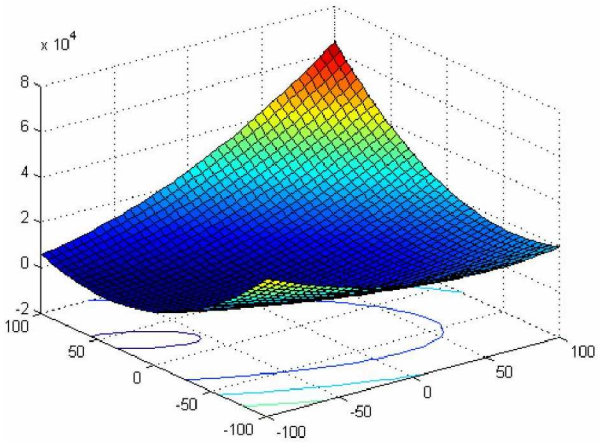


Figura A.3: Función Schwefel 1.2.

$f4$ **- Schwefel 2.21**: función unimodal y separable.

$$f4(x) = max_i\{|x_i|, 1 \leq i \leq 30\}$$

$$-100 \leq x_i \leq 100 \qquad\qquad min(f4) = f4(0, \cdots, 0) = 0$$

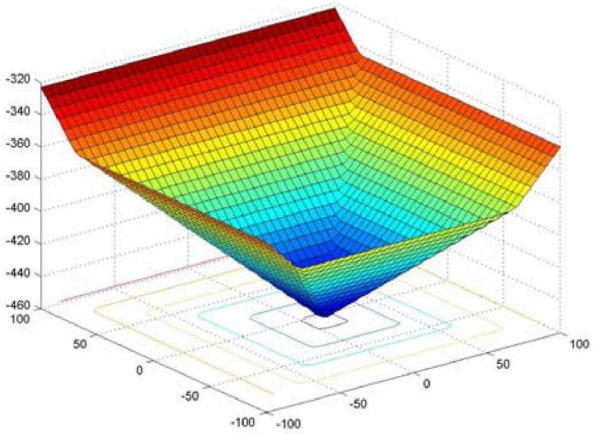


Figura A.4: Función Schwefel 2.21.

$f5$ **- Rosenbrock Generalizada**: función multimodal y no separable.

$$f5(x) = \sum_{i=1}^{29} |100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2|$$

$$-30 \leq x_i \leq 30 \qquad\qquad min(f5) = f5(1, \cdots, 1) = 0$$

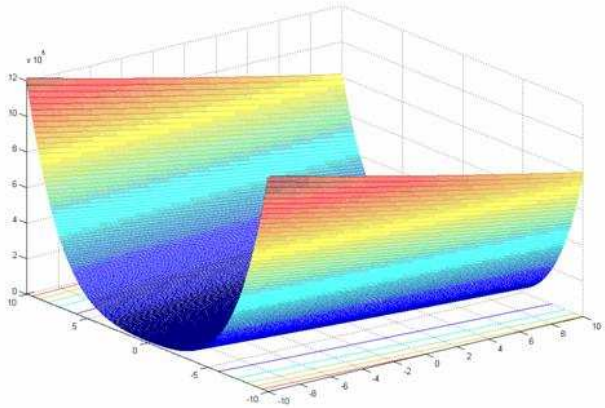


Figura A.5: Función Rosenbrock Generalizada.

$f6$ **- Función de Paso**: unimodal y separable.

$$f6(x) = \sum_{i=1}^{30} (\lfloor x_i + 0{,}5 \rfloor)^2$$

$$-100 \leq x_i \leq 100 \qquad\qquad min(f6) = f6(0, \cdots, 0) = 0$$

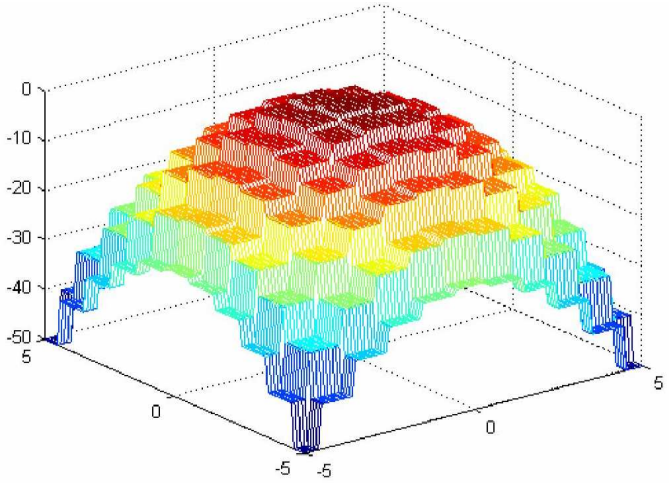


Figura A.6: Función de Paso.

$f7$ **- Cuadrática con ruido**: función unimodal y separable.

$$f7(x) = \sum_{i=1}^{30} i x_i^4 + random[0, 1)$$

$$-1{,}28 \leq x_i \leq 1{,}28 \qquad\qquad min(f7) = f7(0, \cdots, 0) = 0$$

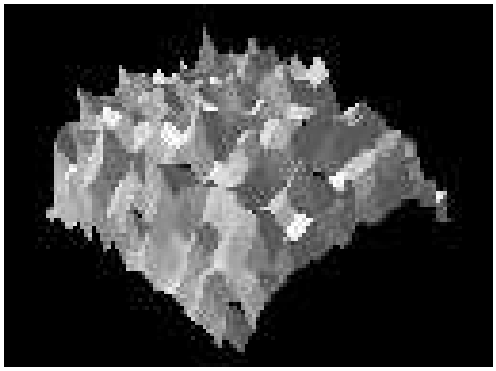


Figura A.7: Función Cuadrática con ruido.

$f8$ **- Schwefel 2.26 Generalizada**: función multimodal y separable con muchos mínimos locales.

$$f8(x) = \sum_{i=1}^{30}(-x_i sin(\sqrt{|x_i|}))$$

$$-500 \leq x_i \leq 500 \qquad min(f8) = f8(420{,}9687, \cdots, 420{,}9687) = -12569{,}48661$$

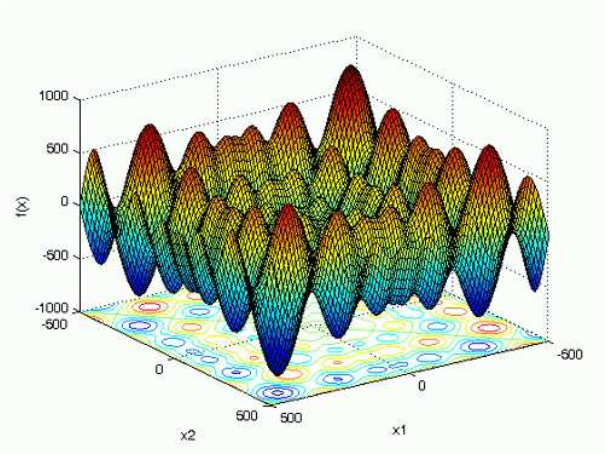


Figura A.8: Función Schwefel 2.26 Generalizada.

$f9$ **- Rastrigin Generalizada** : función multimodal y separable con muchos mínimos locales.

$$f9(x) = \sum_{i=1}^{30}[(x_i^2 - 10cos(2\pi x_i) + 10]$$

$$-5{,}12 \leq x_i \leq 5{,}12 \qquad min(f9) = f9(0, \cdots, 0) = 0$$

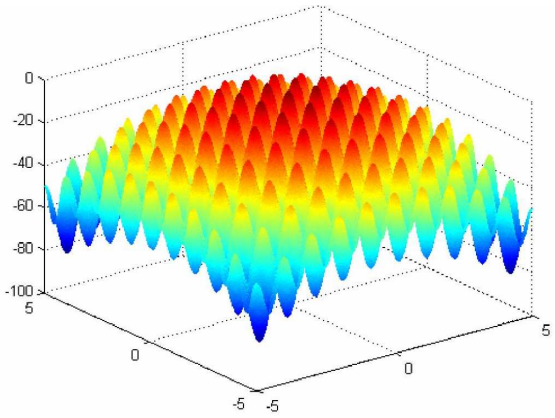


Figura A.9: Función Rastrigin Generalizada.

$f10$ **- Ackley**: función multimodal y no separable con muchos mínimos locales.

$$f10(x) = -20exp\left(-0{,}2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right) - exp\left(\frac{1}{30}\sum_{i=1}^{30} cos(2\pi x_i)\right) + 20 + e$$

$$-32 \leq x_i \leq 32 \qquad\qquad min(f10) = f10(0, \cdots, 0) = 0$$

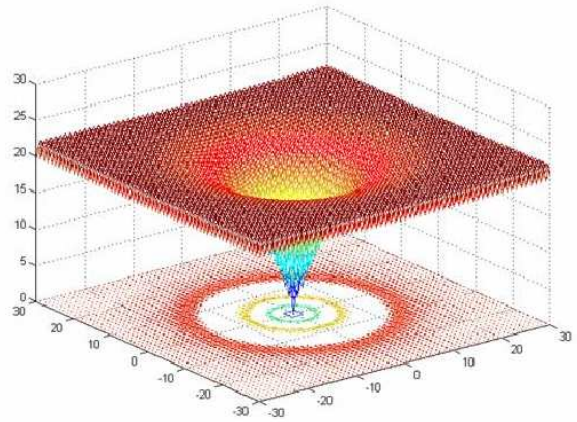


Figura A.10: Función Ackley.

$f11$ **- Griewank Generalizada**: función multimodal y no separable con muchos mínimos locales.

$$f11(x) = \frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$-600 \leq x_i \leq 600 \qquad\qquad min(f11) = f11(0, \cdots, 0) = 0$$

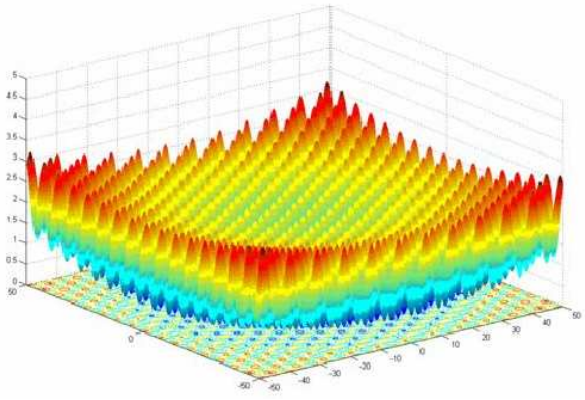


Figura A.11: Función Griewank Generalizada.

$f12$ **- Penalizada Generalizada**: función multimodal y no separable con muchos mínimos locales.

$$f12(x) = \frac{\pi}{30}\left\{10sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i-1)^2[1+10sin^2(\pi y_{1+1})] + (y_n-1)^2\right\}$$

$$+\sum_{i=1}^{30} u(x_i, 10, 100, 4)$$

$$y_i = 1 + \frac{1}{4}(x_i+1)$$

$$u(x_i,a,k,m) = \begin{cases} k(x_i-a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i-a)^m & x_i < -a \end{cases}$$

$$-50 \le x_i \le 50 \qquad\qquad min(f12) = f12(0,\cdots,0) = 0$$

$f13$ **- Penalizada Generalizada 2**: función multimodal y no separable con muchos mínimos locales.

$$f13(x) = 0{,}1\left\{sin^2(3\pi x_1) + \sum_{i=1}^{29}(x_i-1)^2[1+sin^2(3\pi x_{1+1})] + (x_n-1)[1+sin^2(2\pi x_n)]\right\}$$

$$+\sum_{i=1}^{30} u(x_i, 5, 100, 4)$$

$$u(x_i,a,k,m) = \begin{cases} k(x_i-a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i-a)^m & x_i < -a \end{cases}$$

$$-50 \le x_i \le 50 \qquad\qquad min(f13) = f13(0,\cdots,0) = 0$$

Apéndice B

Funciones mono-objetivo con restricciones

g1:

$$f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \qquad sujeto\ a:$$

$$\begin{gathered} g_1(\vec{x}) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\ g_2(\vec{x}) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\ g_3(\vec{x}) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\ g_4(\vec{x}) = -8x_1 + x_{10} \leq 0 \\ g_5(\vec{x}) = -8x_2 + x_{11} \leq 0 \\ g_6(\vec{x}) = -8x_3 + x_{12} \leq 0 \\ g_7(\vec{x}) = -2x_4 - x_5 + x_{10} \leq 0 \\ g_8(\vec{x}) = -2x_6 - x_7 + x_{11} \leq 0 \\ g_9(\vec{x}) = -2x_8 - x_9 + x_{12} \leq 0 \end{gathered}$$

con $0 \leq x_i \leq 1$ $(i = 1, ..., 9)$, $0 \leq x_i \leq 100$ $(i = 10, 11, 12)$ y $0 \leq x_{13} \leq 1$. El óptimo global es x^* =(1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1) y $f(x^*)$ = -15.000. Las restricciones g_1, g_2, g_3, g_7, g_8 y g_9 están activas en el óptimo.

g2:

$$f(\vec{x}) = -\left|\frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_1^2}}\right| \qquad sujeto\ a:$$

$$g_1(\vec{x}) = 0{,}75 - \prod_{i=1}^{n} x_i \leq 0$$

$$g_2(\vec{x}) = \sum_{i=1}^{n} x_i - 7{,}5n \leq 0$$

con $n = 20$ y $0 \leq x_i \leq 10$ $(i = 1, ..., n)$. El óptimo global se desconoce. La mejor solución conocida es $f(x^*)$ = -0.803. La restricción g_1 está casi activa ($g_1 = -10^{-8}$).

g3:

$$f(\vec{x}) = -\left(\sqrt{n}\right)^n \prod_{i=1}^{n} x_i \qquad sujeto\ a:$$

$$h(\vec{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

con $n = 10$ y $0 \leq x_i \leq 1$ $(i = 1, ..., n)$. El óptimo global es x^* =(0.3162, 0.3162, 0.3162, 0.3162, 0.3162, 0.3162, 0.3162, 0.3162, 0.3162, 0.3162) y $f(x^*)$ = -1.000.

g4:

$$f(\vec{x}) = 5{,}3578547x_3^2 + 0{,}8356891x_1x_5 + 37{,}293239x_1 - 40,792{,}141 \qquad sujeto\ a:$$

$$\begin{gathered}
g_1(\vec{x}) = 85{,}334407 + 0{,}0056858x_2x_5 + 0{,}0006262x_1x_4 - 0{,}0022053x_3x_5 - 92 \leq 0 \\
g_2(\vec{x}) = -85{,}334407 - 0{,}0056858x_2x_5 - 0{,}0006262x_1x_4 + 0{,}0022053x_3x_5 \leq 0 \\
g_3(\vec{x}) = 80{,}51249 + 0{,}0071317x_2x_5 + 0{,}0029955x_1x_2 + 0{,}0021813x_3^2 - 110 \leq 0 \\
g_4(\vec{x}) = -80{,}51249 - 0{,}0071317x_2x_5 - 0{,}0029955x_1x_2 - 0{,}0021813x_3^2 + 90 \leq 0 \\
g_5(\vec{x}) = 9{,}300961 + 0{,}0047026x_3x_5 + 0{,}0012547x_1x_3 + 0{,}0019085x_3x_4 - 25 \leq 0 \\
g_6(\vec{x}) = -9{,}300961 - 0{,}0047026x_3x_5 - 0{,}0012547x_1x_3 - 0{,}0019085x_3x_4 + 20 \leq 0
\end{gathered}$$

con $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$ y $27 \leq x_i \leq 45$ $(i = 3, 4, 5)$. El óptimo global es x^* =(78, 33, 29.9952, 45, 36.7758) y $f(x^*)$ = -30,665.539. Las restricciones g_1 y g_6 están activas en el óptimo.

g5:

$$f(\vec{x}) = 3x_1 + 0{,}000001x_1^3 + 2x_2 + (0{,}000002/3)x_2^3 \qquad sujeto\ a:$$

$$\begin{gathered}
g_1(\vec{x}) = -x_4 + x_3 - 0{,}55 \leq 0 \\
g_2(\vec{x}) = -x_3 + x_4 - 0{,}55 \leq 0 \\
h_3(\vec{x}) = 1,000\sin(-x_3 - 0{,}25) + 1,000\sin(-x_4 - 0{,}25) + 894{,}8 - x_1) = 0 \\
h_4(\vec{x}) = 1,000\sin(x_3 - 0{,}25) + 1,000\sin(x_3 - x_4 - 0{,}25) + 894{,}8 - x_2) = 0 \\
h_5(\vec{x}) = 1,000\sin(x_4 - 0{,}25) + 1,000\sin(x_4 - x_3 - 0{,}25) + 1,294{,}8 = 0
\end{gathered}$$

con $0 \leq x_i \leq 1,200$ $(i = 1, 2)$ y $-0{,}55 \leq x_i \leq 0{,}55$ $(i = 3, 4)$. El óptimo global es x^* =(679.9451, 1,026.0669, 0.1188, -0.3962) y $f(x^*)$ = 5,126.496.

g6:

$$f(\vec{x}) = (x_1 - 10)^3 + (x_2 - 20)^3 \qquad sujeto\ a:$$

$$\begin{gathered}
g_1(\vec{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\
g_2(\vec{x}) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82{,}81 \leq 0
\end{gathered}$$

con $13 \leq x_1 \leq 100$ y $0 \leq x_2 \leq 100$. El óptimo global es $x^* = (14.0950, 0.8429)$ y $f(x^*) =$ -6,961.813. Ambas restricciones están activas en el óptimo.

g7:

$$f(\vec{x}) = x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2$$
$$+(x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2$$
$$+7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \qquad sujeto\ a:$$

$$g_1(\vec{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$$
$$g_2(\vec{x}) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0$$
$$g_3(\vec{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$$
$$g_4(\vec{x}) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$$
$$g_5(\vec{x}) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0$$
$$g_6(\vec{x}) = x_1^2 + 2(x_2 - 2)^2 - 2x_1 x_2 + 14x_5 - 6x_6 \leq 0$$
$$g_7(\vec{x}) = 0{,}5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$$
$$g_8(\vec{x}) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$$

con $-10 \leq x_i \leq 10$ $(i = 1, ..., 10)$. El óptimo global es x^* =(2.1719, 2.3636, 8.7739, 5.0959, 0.9906, 1.4305, 1.3216, 9.8287, 8.2800, 8.3759) y $f(x^*) = 24.306$. Las restricciones g_1, g_2, g_3, g_4, g_5 y g_6 están activas en el óptimo.

g8:

$$f(\vec{x}) = -\frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)} \qquad sujeto\ a:$$

$$g_1(\vec{x}) = x_1^2 - x_2 + 1 \leq 0$$
$$g_2(\vec{x}) = 1 - x_1 + (x_2 - 4)^2 \leq 0$$

con $0 \leq x_1 \leq 10$ y $0 \leq x_2 \leq 10$. El óptimo global es $x^* = (1.2279, 4.2453)$ y $f(x^*) =$ -0.095.

g9:

$$f(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2$$
$$+10x_5^6 + 7x_6^2 + x_7^4 - 4x_6 x_7 - 10x_6 - 8x_7 \qquad sujeto\ a:$$

$$g_1(\vec{x}) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$$
$$g_2(\vec{x}) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0$$
$$g_3(\vec{x}) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$$
$$g_4(\vec{x}) = 4x_1^2 + x_2^2 - 3x_1 x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$$

con $-10 \leq x_i \leq 10$ $(i = 1, ..., 7)$. El óptimo global es x^* =(2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227) y $f(x^*) = 680.630$. Las restricciones g_1 y g_4 están activas en el óptimo.

g10:

$$f(\vec{x}) = x_1 + x_2 + x_3 \qquad sujeto\ a:$$

$$g_1(\vec{x}) = -1 + 0{,}0025(x_4 + x_6) \leq 0$$
$$g_2(\vec{x}) = -1 + 0{,}0025(x_5 + x_7 - x_4) \leq 0$$
$$g_3(\vec{x}) = -1 + 0{,}01(x_8 - x_5) \leq 0$$
$$g_4(\vec{x}) = -x_1x_6 + 833{,}33252x_4 + 100x_1 - 83,333{,}333 \leq 0$$
$$g_5(\vec{x}) = -x_2x_7 + 1,250x_5 + x_2x_4 - 1,250x_4 \leq 0$$
$$g_6(\vec{x}) = -x_3x_8 + 1,250,000 + x_3x_5 - 2,500x_5 \leq 0$$

con $100 \leq x_1 \leq 10,000$, $1,000 \leq x_i \leq 10,000$ $(i = 2, 3)$ y $10 \leq x_i \leq 1,000$ $(i = 4, ..., 8)$. El óptimo global es x^* =(579.3066, 1,359.9706, 5,109.9706, 182.0176, 295.6011, 217.9823, 286.4165, 395.6011) y $f(x^*) = 7{,}049.248$. Las restricciones g_1, g_2 y g_3 están activas en el óptimo.

g11:

$$f(\vec{x}) = x_1^2 + (x_2 - 1)^2 \qquad sujeto\ a:$$

$$h(\vec{x}) = x_2 - x_1^2 = 0$$

con $-1 \leq x_i \leq 1$ $(i = 1, 2)$. El óptimo global es x^* =(-0.7070, 0.5000) y $f(x^*) = 0.749$.

g12:

$$f(\vec{x}) = -\frac{(100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2}{100} \qquad sujeto\ a:$$

$$g(\vec{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0{,}0625 \leq 0$$

con $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$ y p, q, r =1,2,...,9. El óptimo global es x^* =(5, 5, 5) y $f(x^*)$ = -1.000.

g13:

$$f(\vec{x}) = e^{x_1x_2x_3x_4x_5} \qquad sujeto\ a:$$

$$h_1(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0$$
$$h_2(\vec{x}) = x_2x_3 - 5x_4x_5 = 0$$
$$h_3(\vec{x}) = x_1^3 + x_2^3 + 1 = 0$$

con $-2{,}3 \leq x_i \leq 2{,}3$ $(i = 1, 2)$ y $-3{,}2 \leq x_i \leq 3{,}2$ $(i = 3, 4, 5)$. El óptimo global es x^* =(-1.7171, 1.5957, 1.8272, -0.7636, -0.7636) y $f(x^*) = 0.053$.

g14:

$$f(\vec{x}) = \sum_{i=1}^{10} x_i \left(c_i + ln \frac{x_i}{\sum_{j=1}^{10} x_j} \right) \qquad sujeto\ a:$$

$$h_1(\vec{x}) = x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 = 0$$
$$h_2(\vec{x}) = x_4 + 2x_5 + x_6 + x_7 - 1 = 0$$
$$h_3(\vec{x}) = x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 = 0$$

con c_1 =-6.089, c_2 =-17.164, c_3 =-34.054, c_4 =-5.914, c_5 =-24.721, c_6 =-14.986, c_7 =-24.100, c_8 =-10.708, c_9 =-26.662 y c_{10} =-22.179. $0 \leq x_i \leq 1$ $(i = 1, ..., 10)$. El óptimo global es x^* = (0.0406, 0.1477, 0.7832, 0.0014, 0.4852, 0.0006, 0.0274, 0.0179, 0.0373, 0.0968) y $f(x^*)$ = -47.764.

g15:

$$f(\vec{x}) = 1,000,0 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3 \qquad sujeto\ a:$$

$$h_1(\vec{x}) = x_1^2 + x_2^2 + x_3^2 - 25,0 = 0$$
$$h_2(\vec{x}) = 8x_1 + 14x_2 + 7x_3 - 56 = 0$$

con $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$. El óptimo global es x^* = (3.5121, 0.2169, 3.5521) y $f(x^*)$ = 961.715.

g16:

$$f(\vec{x}) = -(0,0000005843y_{17} - 0,000117y_{14} - 0,1365 - 0,00002358y_{13}$$

$$-0,000001502y_{16} - 0,0321y_{12} - 0,004324y_5 - 0,0001\frac{c_{15}}{c_{16}} - 37,48\frac{y_2}{c_{12}}) \qquad donde:$$

$$y_1 = x_2 + x_3 + 41{,}6$$
$$c_1 = 0{,}024x_4 - 4{,}62$$
$$y_2 = \frac{12{,}5}{c_1} + 12{,}0$$
$$c_2 = 0{,}0003535x_1^2 + 0{,}5311x_1 + 0{,}08705y_2x_1$$
$$c_3 = 0{,}052x_1 + 78{,}0 + 0{,}002377y_2x_1$$
$$y_3 = \frac{c_2}{c_3}$$
$$y_4 = 19y_3$$
$$c_4 = 0{,}04782(x_1 - y_3) + \frac{0{,}1956(x_1 - y_3)^2}{x_2} + 0{,}6376y_4 + 1{,}594y_3$$
$$c_5 = 100x_2$$
$$c_6 = x_1 - y_3 - y_4$$
$$c_7 = 0{,}950 - \frac{c_4}{c_5}$$
$$y_5 = c_6c_7$$
$$y_6 = x_1 - y_5 - y_4 - y_3$$
$$c_8 = (y_5 + y_4)0{,}995$$
$$y_7 = \frac{c_8}{y_1}$$
$$y_8 = \frac{c_8}{3{,}798{,}0}$$
$$c_9 = y_7 - \frac{0{,}0663y_7}{y_8} - 0{,}3153$$
$$y_9 = \frac{96{,}82}{c_9} + 0{,}321y_1$$
$$y_{10} = 1{,}29y_5 + 1{,}258y_4 + 2{,}29y_3 + 1{,}71y_6$$
$$y_{11} = 1{,}71x_1 - 0{,}452y_4 + 0{,}580y_3$$
$$c_{10} = \frac{12{,}3}{752{,}3}$$
$$c_{11} = (1{,}75y_2)(0{,}995x_1)$$
$$c_{12} = 0{,}995y_{10} + 1{,}998{,}0$$
$$y_{12} = c_{10}x_1 + \frac{c_{11}}{c_{12}}$$
$$y_{13} = c_{12} - 1{,}75y_2$$
$$y_{14} = 3{,}623{,}0 + 64{,}4x_2 + 58{,}4x_3 + \frac{146{,}312{,}0}{y_9 + x_5}$$
$$c_{13} = 0{,}995y_{10} + 60{,}8x_2 + 48x_4 - 0{,}1121y_{12} - 5{,}095{,}0$$
$$y_{15} = \frac{y_{13}}{c_{13}}$$
$$y_{16} = 148{,}000{,}0 - 331{,}000{,}0y_{15} + 40{,}0y_{13} - 61{,}0y_{15}y_{13}$$
$$c_{14} = 2{,}324y_{10} - 28{,}740{,}000y_2$$
$$y_{17} = 14{,}130{,}000 - 1{,}328{,}0y_{10} - 531{,}0y_{11} + \frac{c_{14}}{c_{12}}$$
$$c_{15} = \frac{y_{13}}{y_{15}} - \frac{y_{13}}{0{,}52}$$
$$c_{16} = 1{,}104 - 0{,}72y_{15}$$
$$c_{17} = y_9 + x_5$$

$$sujeto\ a:$$

$$g_1(\vec{x}) = y_4 - \frac{0{,}28}{0{,}72} y_5 \geq 0$$
$$g_2(\vec{x}) = 1{,}5x_2 - x_3 \geq 0$$
$$g_3(\vec{x}) = 21 - 3{,}496\frac{y_2}{c_{12}} \geq 0$$
$$g_4(\vec{x}) = \frac{62{,}212{,}0}{c_{17}} - 110{,}6 - y_1 \geq 0$$
$$g_5(\vec{x}) = 213{,}1 \leq y_1 \leq 405{,}23$$
$$g_6(\vec{x}) = 17{,}505 \leq y_2 \leq 1{,}053{,}6667$$
$$g_7(\vec{x}) = 11{,}275 \leq y_3 \leq 35{,}03$$
$$g_8(\vec{x}) = 214{,}228 \leq y_4 \leq 665{,}585$$
$$g_9(\vec{x}) = 7{,}458 \leq y_5 \leq 584{,}463$$
$$g_{10}(\vec{x}) = 0{,}961 \leq y_6 \leq 265{,}916$$
$$g_{11}(\vec{x}) = 1{,}612 \leq y_7 \leq 7{,}046$$
$$g_{12}(\vec{x}) = 0{,}146 \leq y_8 \leq 0{,}222$$
$$g_{13}(\vec{x}) = 107{,}99 \leq y_9 \leq 273{,}366$$
$$g_{14}(\vec{x}) = 922{,}693 \leq y_{10} \leq 1{,}286{,}105$$
$$g_{15}(\vec{x}) = 926{,}832 \leq y_{11} \leq 1{,}444{,}046$$
$$g_{16}(\vec{x}) = 18{,}766 \leq y_{12} \leq 537{,}141$$
$$g_{17}(\vec{x}) = 1{,}072{,}163 \leq y_{13} \leq 3{,}247{,}039$$
$$g_{18}(\vec{x}) = 8{,}961{,}448 \leq y_{14} \leq 26{,}844{,}086$$
$$g_{19}(\vec{x}) = 0{,}063 \leq y_{15} \leq 0{,}386$$
$$g_{20}(\vec{x}) = 71{,}084{,}33 \leq y_{16} \leq 140{,}000{,}0$$
$$g_{21}(\vec{x}) = 2{,}802{,}713{,}0 \leq y_{17} \leq 12{,}146{,}108{,}0$$

con $704{,}4148 \leq x_1 \leq 906{,}3855$, $68{,}6 \leq x_2 \leq 288{,}88$, $0 \leq x_3 \leq 134{,}75$, $193 \leq x_4 \leq 287{,}0966$ y $25 \leq x_5 \leq 84{,}1988$. El óptimo global es $x^* =$ (705.1745, 68.5999, 102.8999, 282.3249, 37.5841) y $f(x^*) =$ -1.905.

g17:

$$f(\vec{x}) = f_1(x_1) + f_2(x_2) \qquad where:$$
$$f_1(x_1) = \left\{ \begin{array}{ll} 30x_1 & 0 \leq x_1 < 300 \\ 31x_1 & 300 \leq x_1 < 400 \end{array} \right\}$$
$$f_2(x_2) = \left\{ \begin{array}{ll} 28x_2 & 0 \leq x_2 < 100 \\ 29x_2 & 100 \leq x_2 < 200 \\ 30x_2 & 200 \leq x_2 < 1{,}000 \end{array} \right\}$$

$$sujeto\ a:$$

$$h_1(\vec{x}) = x_1 - 300 + \frac{x_3 x_4}{131{,}078}\cos(1{,}48477 - x_6) - \frac{0{,}90798}{131{,}078}x_3^2\cos(1{,}47588) = 0$$
$$h_2(\vec{x}) = x_2 + \frac{x_3 x_4}{131{,}078}\cos(1{,}48477 + x_6) - \frac{0{,}90798}{131{,}078}x_4^2\cos(1{,}47588) = 0$$
$$h_3(\vec{x}) = x_5 + \frac{x_3 x_4}{131{,}078}\sin(1{,}48477 + x_6) - \frac{0{,}90798}{131{,}078}x_4^2\sin(1{,}47588) = 0$$
$$h_4(\vec{x}) = 200 - + \frac{x_3 x_4}{131{,}078}\sin(1{,}48477 - x_6) + \frac{0{,}90798}{131{,}078}x_3^2\sin(1{,}47588) = 0$$

con $0 \leq x_1 \leq 400$, $0 \leq x_2 \leq 1{,}000$, $340 \leq x_3 \leq 420$, $340 \leq x_4 \leq 420$, $-1{,}000 \leq x_5 \leq 1{,}000$ y $0 \leq x_6 \leq 0{,}5236$. El óptimo global es $x^* =$(201.7844, 99.9999, 383.0710, 420,

-10.9076, 0.07314) y $f(x^*) = 8{,}853.539$.

g18:

$$f(\vec{x}) = -0{,}5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7) \qquad sujeto\ a:$$

$$g_1(\vec{x}) = 1 - x_3^2 - x_4^2 \geq 0$$
$$g_2(\vec{x}) = 1 - x_9^2 \geq 0$$
$$g_3(\vec{x}) = 1 - x_5^2 - x_6^2 \geq 0$$
$$g_4(\vec{x}) = 1 - x_1^2 - (x_2 - x_9)^2 \geq 0$$
$$g_5(\vec{x}) = 1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 \geq 0$$
$$g_6(\vec{x}) = 1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 \geq 0$$
$$g_7(\vec{x}) = 1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \geq 0$$
$$g_8(\vec{x}) = 1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \geq 0$$
$$g_9(\vec{x}) = 1 - x_7^2 - (x_8 - x_9)^2 \geq 0$$
$$g_{10}(\vec{x}) = x_1x_4 - x_2x_3 \geq 0$$
$$g_{11}(\vec{x}) = x_3x_9 \geq 0$$
$$g_{12}(\vec{x}) = -x_5x_9 \geq 0$$
$$g_{13}(\vec{x}) = x_5x_8 - x_6x_7 \geq 0$$

con $-1 \leq x_i \leq 1$ $(i = 1, ..., 8)$ y $0 \leq x_9 \leq 1$. El óptimo global es x^* =(-0.6577, -0.1534, 0.3234, -0.9462, -0.6577, -0.7532, 0.3234, -0.3464, 0.5997) y $f(x^*) = -0.866$.

g19:

$$f(\vec{x}) = -(\sum_{i=1}^{10} b_i x_i - \sum_{j=1}^{5}\sum_{i=1}^{5} c_{ij}x_{(10+i)}x_{(10+j)} - 2\sum_{j=1}^{5} d_j x_{(10+j)}^3) \qquad sujeto\ a:$$

$$g_j(\vec{x}) = 2\sum_{i=1}^{5} c_{ij}x_{(10+i)} + 3d_j x_{(10+j)}^2$$

$$+e_j - \sum_{i=1}^{10} a_{ij}x_i \geq 0 \ \ j = 1, ..., 5$$

con b_1 =-40, b_2 =-2, b_3 =-0.25, b_4 =-4, b_5 =-4, b_6 =-1, b_7 =-40, b_8 =-60, b_9 =5, b_{10} =1 y e_j, c_{ij}, d_j, a_{ij} se especifican en la tabla B.1. $0 \leq x_i \leq 10$ $(i = 1, ..., 15)$. El óptimo global es x^* = (0, 0, 3.9459, 0, 3.2831, 9.9999, 0, 0, 0, 0, 0.3707, 0.2784, 0.5238, 0.3886, 0.2981) y $f(x^*) = 32.655$.

g20:

$$f(\vec{x}) = \sum_{i=1}^{24} a_i x_i \qquad sujeto\ a:$$

$$h_i(\vec{x}) = \frac{x_{(i+12)}}{b_{(i+12)}\sum_{j=13}^{24}\frac{x_j}{b_j}} - \frac{c_i x_i}{40b_i\sum_{j=1}^{12}\frac{x_j}{b_j}} = 0 \ \ j = 1, ..., 12$$

Tabla B.1: Datos para g19.

j	1	2	3	4	5
e_j	-15.0	-27.0	-36.0	-18.0	-12.0
c_{1j}	30.0	-20.0	-10.0	32.0	-10.0
c_{2j}	-20.0	39.0	-6.0	-31.0	32.0
c_{3j}	-10.0	-6.0	10.0	-6.0	-10.0
c_{4j}	32.0	-31.0	-6.0	39.0	-20.0
c_{5j}	-10.0	32.0	-10.0	-20.0	30.0
d_j	4.0	8.0	10.0	6.0	2.0
a_{1j}	-16.0	2.0	0.0	1.0	0.0
a_{2j}	0.0	-2.0	0.0	0.4	2.0
a_{3j}	-3.5	0.0	2.0	0.0	0.0
a_{4j}	0.0	-2.0	0.0	-4.0	-1.0
a_{5j}	0.0	-9.0	-2.0	1.0	-2.8
a_{6j}	2.0	0.0	-4.0	0.0	0.0
a_{7j}	-1.0	-1.0	-1.0	-1.0	-1.0
a_{8j}	-1.0	-2.0	-3.0	-2.0	-1.0
a_{9j}	1.0	2.0	3.0	4.0	5.0
a_{10j}	1.0	1.0	1.0	1.0	1.0

$$h_{13}(\vec{x}) = \sum_{i=1}^{24} x_i - 1 = 0$$

$$h_{14}(\vec{x}) = \sum_{i=1}^{12} \frac{x_i}{d_j} + (0{,}7302)(530)\left(\frac{14{,}7}{40}\right)\sum_{i=13}^{24} \frac{x_j}{b_j} - 1{,}671 = 0$$

$$g_i(\vec{x}) = -\frac{x_i + x_{(i+12)}}{\sum_{j=1}^{24} x_j + e_i} \geq 0 \;\; i = 1, 2, 3$$

$$g_k(\vec{x}) = -\frac{x_{(k+3)} + x_{(k+15)}}{\sum_{j=1}^{24} x_j + e_k} \geq 0 \;\; i = 4, 5, 6$$

con e_i, c_i, d_i, b_i, a_i se especifican en la tabla B.2 y $0 \leq x_i \leq 1$ $(i = 1, ..., 24)$. La mejor solución conocida posee valor objetivo $f(x^*) = 0.097$ pero es infactible.

g21:

$$f(\vec{x}) = x_1 \qquad sujeto\ a:$$

$$g_1(\vec{x}) = -x_1 + 35{x_2}^{0,6} + 35{x_3}^{0,6} \leq 0$$

$$h_1(\vec{x}) = -300x_3 + 7,500x_5 - 7,500x_6 - 25x_4x_5 + 25x_4x_6 + x_3x_4 = 0$$

$$h_2(\vec{x}) = 100x_2 + 155{,}365x_4 + 2,500x_7 - x_2x_4 - 25x_4x_7 - 15,536{,}5 = 0$$

$$h_3(\vec{x}) = -x_5 + ln(-x_4 + 900) = 0$$

Tabla B.2: Datos para g20.

i	a_i	b_i	c_i	d_i	e_i
1	0.0693	44.094	123.7	31.244	0.1
2	0.0577	58.12	31.7	36.12	0.3
3	0.05	58.12	45.7	34.784	0.4
4	0.2	137.4	14.7	92.7	0.3
5	0.26	120.9	84.7	82.7	0.6
6	0.55	170.9	27.7	91.6	0.3
7	0.06	62.501	49.7	56.708	
8	0.1	84.94	7.1	82.7	
9	0.12	133.425	2.1	80.8	
10	0.18	82.507	17.7	64.517	
11	0.1	46.07	0.85	49.4	
12	0.09	60.097	0.64	49.1	
13	0.0693	44.094			
14	0.0577	58.12			
15	0.05	58.12			
16	0.2	137.4			
17	0.26	120.9			
18	0.55	170.9			
19	0.06	62.501			
20	0.1	84.94			
21	0.12	133.425			
22	0.18	82.507			
23	0.1	46.07			
24	0.09	60.097			

$$h_4(\vec{x}) = -x_6 + ln(x_4 + 300) = 0$$

$$h_5(\vec{x}) = -x_7 + ln(-2x_4 + 700) = 0$$

con $0 \leq x_1 \leq 1,000$, $0 \leq x_2, x_3 \leq 40$, $100 \leq x_4 \leq 300$, 6,30 $\leq x_5 \leq$ 6,7, 5,9 $\leq x_6 \leq$ 6,4 y 4,5 $\leq x_7 \leq$ 6,25. El óptimo global es $x^* =$ (193.724510070034967, 5.56944131553368433e-27, 17.3191887294084914, 100.047897801386839, 6.68445185362377892, 5.99168428444264833, 6.21451648886070451) y $f(x^*) = 193.724$.

g22:

$$f(\vec{x}) = x_1 \qquad sujeto\ a:$$

$$g_1(\vec{x}) = -x_1 + {x_2}^{0,6} + {x_3}^{0,6} + {x_4}^{0,6} \leq 0$$

$$h_1(\vec{x}) = x_5 - 100,000x_8 + 1 \times 10^7 = 0$$

$$h_2(\vec{x}) = x_6 + 100,000x_8 - 100,000x_9 = 0$$

$$h_3(\vec{x}) = x_7 + 100,000x_9 - 5 \times 10^7 = 0$$
$$h_4(\vec{x}) = x_5 + 100,000x_{10} - 3{,}3 \times 10^7 = 0$$
$$h_5(\vec{x}) = x_6 + 100,000x_{11} - 4{,}4 \times 10^7 = 0$$
$$h_6(\vec{x}) = x_7 + 100,000x_{12} - 6{,}6 \times 10^7 = 0$$
$$h_7(\vec{x}) = x_5 - 120x_2x_{13} = 0$$
$$h_8(\vec{x}) = x_6 - 80x_3x_{14} = 0$$
$$h_9(\vec{x}) = x_7 - 40x_4x_{15} = 0$$
$$h_{10}(\vec{x}) = x_8 - x_{11} + x_{16} = 0$$
$$h_{11}(\vec{x}) = x_9 - x_{12} + x_{17} = 0$$
$$h_{12}(\vec{x}) = -x_{18} + ln(x_{10} - 100) = 0$$
$$h_{13}(\vec{x}) = -x_{19} + ln(-x_8 + 300) = 0$$
$$h_{14}(\vec{x}) = -x_{20} + ln(x_{16}) = 0$$
$$h_{15}(\vec{x}) = -x_{21} + ln(-x_9 + 400) = 0$$
$$h_{16}(\vec{x}) = -x_{22} + ln(x_{17}) = 0$$
$$h_{17}(\vec{x}) = -x_8 - x_{10} + x_{13}x_{18} - x_{13}x_{19} + 400 = 0$$
$$h_{18}(\vec{x}) = x_8 - x_9 - x_{11} + x_{14}x_{20} - x_{14}x_{21} + 400 = 0$$
$$h_{19}(\vec{x}) = x_9 - x_{12} - 4{,}60517x_{15} + x_{15}x_{22} + 100 = 0$$

con $0 \leq x_1 \leq 20,000$, $0 \leq x_2, x_3, x_4 \leq 1 \times 10^6$, $0 \leq x_5, x_6, x_7 \leq 4 \times 10^7$, $100 \leq x_8 \leq 299{,}99$, $100 \leq x_9 \leq 399{,}99$, $100{,}01 \leq x_{10} \leq 300$, $100 \leq x_{11} \leq 400$, $100 \leq x_{12} \leq 600$, $0 \leq x_{13}, x_{14}, x_{15} \leq 500$, $0{,}01 \leq x_{16} \leq 300$, $0{,}01 \leq x_{17} \leq 400$ y $-4{,}7 \leq x_{18}, x_{19}, x_{20}, x_{21}, x_{22} \leq 6{,}25$. El óptimo global es $x^* =$ (236.4309, 135.8284, 204.8181, 6,446.5465, 3,007,540.8394, 4,074,188.6577, 32,918,270.5028, 130.0754, 170.8172, 299.9245, 399.2581, 330.8172, 184.5183, 248.6467, 127.6585, 269.1826, 160.00001, 5.2978, 5.13529, 5.59531, 5.43444, 5.0751) y $f(x^*) = 236.430975504001$.

g23:

$$f(\vec{x}) = -9x_5 - 15x_8 + 6x_1 + 16x_2 + 10(x_6 + x_7) \qquad sujeto\ a:$$
$$g_1(\vec{x}) = x_9x_3 + 0{,}02x_6 - 0{,}025x_5 \leq 0$$
$$g_2(\vec{x}) = x_9x_4 + 0{,}02x_7 - 0{,}015x_8 \leq 0$$
$$h_1(\vec{x}) = x_1 + x_2 - x_3 - x_4 = 0$$
$$h_2(\vec{x}) = 0{,}03x_1 + 0{,}01x_2 - x_9(x_3 + x_4) = 0$$
$$h_3(\vec{x}) = x_3 + x_6 - x_5 = 0$$
$$h_4(\vec{x}) = x_4 + x_7 - x_8 = 0$$

con $0 \leq x_1, x_2, x_6 \leq 300$, $0 \leq x_3, x_5, x_7 \leq 100$, $0 \leq x_4, x_8 \leq 200$ y $0{,}01 \leq x_9 \leq 0{,}03$. El óptimo global es $x^* = (0.0051000000000259465, 99.9947000000000514, 9.01920162996045897e\text{-}18, 99.9999000000000535, 0.000100000000027086086, 2.75700683389584542e\text{-}14, 99.9999999999999574, 2{,}000.0100000100000100008)$ y $f(x^*) =$ -400.055099999999584.

g24:

$$f(\vec{x}) = -x_1 - x_2 \qquad sujeto\ a:$$

$$g_1(\vec{x}) = -2{x_1}^4 + 8{x_1}^3 - 8{x_1}^2 + x_2 - 2 \leq 0$$

$$g_2(\vec{x}) = -4{x_1}^4 + 32{x_1}^3 - 88{x_1}^2 + 96x_1 + x_2 - 36 \leq 0$$

con $0 \leq x_1 \leq 3$ y $0 \leq x_2 \leq 4$. El óptimo global es $x^* =$ (2.3295, 3.1784) y $f(x^*) =$ -5.50801327159536.

Apéndice C

Funciones de diseño óptimo de armaduras

Diseño de una armadura plana de 10 barras:

El problema consiste en encontrar la sección transversal de cada barra de forma tal que se minimice el peso de la estructura, sujeto a restricciones de desplazamiento y esfuerzo. El peso de la armadura es una función $f(x)$ definida en la ecuación (C.1). La figura C.1 ilustra este problema [10].

$$f(x) = \sum_{j=1}^{10} \rho A_j L_j \tag{C.1}$$

donde x es la solución propuesta, A_j es la sección transversal, L_j la longitud de la barra j y, ρ la densidad del material. El problema fue modelado utilizando 20 variables de diseño (alto y ancho de cada barra). Se utilizan los siguientes datos:

- Módulo de elasticidad: $E = 1{,}09 \times 10^4$ ksi.
- $\rho = 0{,}10$ lb/in^3.
- Una carga de 100 kips en la dirección negativa de y es aplicada en los nodos 2 y 4.
- Esfuerzo máximo de cada barra: $\sigma_a = \pm 25$ ksi.
- Desplazamiento máximo permitido de cada nodo (horizontal y vertical): $u_a = 2$ in.
- Mínima sección transversal permitida para todas las barras: 0.10 in^2.
- Espesores del entramado y reborde permanecen fijos en 0.1 in.

Para evaluar la función objetivo de este problema se agregó un módulo para el análisis de la armadura [56] que retorna los valores de esfuerzo y desplazamiento así como el peso total de la estructura.

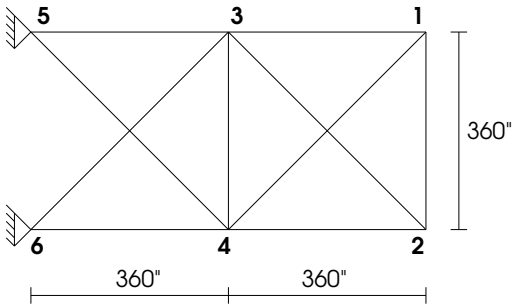


Figura C.1: Armadura plana de 10 barras.

Diseño de una armadura espacial de 25 barras:

La armadura espacial (3D) de 25 barras es mostrada en la figura C.2. El problema consiste en encontrar el área de la sección transversal de cada barra de forma tal que se minimice el peso total de la estructura considerando desplazamientos y esfuerzos máximos permitidos para cada barra [149]. Las condiciones de carga son mostradas en la tabla C.1, las coordenadas de cada nodo se muestran en la tabla C.2 y los grupos de elementos se muestran en tabla C.3. Se utilizan los siguientes datos:

- Módulo de elasticidad: $E = 1 \times 10^7$ ksi.
- Densidad del material $\rho = 0{,}1$ lb/in^3.
- Esfuerzo máximo permitido para cada barra: $\sigma = \pm 40$ ksi.
- Desplazamiento máximo permitido: $u = \pm 0{,}35$ in.

El problema fue modelado usando 8 variables de diseño (una para cada grupo), 25 restricciones de esfuerzo y 18 restricciones de desplazamiento. El peso de la estructura es una función $f(x)$ definida en la ecuación (C.2).

$$f(x) = \sum_{j=1}^{25} \rho A_j L_j \tag{C.2}$$

donde x es la solución propuesta, A_j el área de la sección transversal, L_j la longitud del miembro j y, ρ la densidad del peso del material.

Diseño de una armadura plana de 200 barras:

El problema consiste en encontrar el área de la sección transversal de cada miembro de la estructura de forma tal que el peso total sea minimizado. Este problema tiene sólo restricciones de esfuerzo [10]. La figura C.3 ilustra el problema.

Se utiliza la siguiente información:

Tabla C.1: Carga de nodos para la armadura espacial de 25 barras.

Nodo	F_x (lb)	F_y (lb)	F_z (lb)
1	1,000	-10,000	-10,000
2	0	-10,000	-10,000
3	500	0	0
4	600	0	0

Tabla C.2: Coordenadas de nodos para la armadura espacial de 25 barras.

Nodo	x (cm)	y (cm)	z (cm)
1	-95.25	0	508
2	95.25	0	508
3	-95.25	95.25	254
4	95.25	95.25	254
5	95.25	-95.25	254
6	-95.25	-95.25	254
7	-254	254	0
8	254	254	0
9	254	-254	0
10	-254	-254	0

Tabla C.3: Agrupación de miembros para la armadura espacial de 25 barras.

Grupo	Miembro
1	1-2
2	1-4, 2-3, 1-5, 2-6
3	2-5, 2-4, 1-3, 1-6
4	3-6, 4-5
5	3-4, 5-6
6	3-10, 6-7, 4-9, 5-8
7	3-8, 4-7, 6-9, 5-10
8	3-7, 4-8, 5-9, 6-10

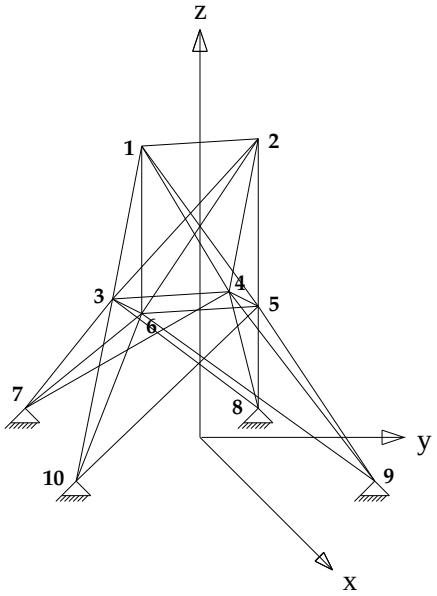


Figura C.2: Armadura espacial de 25 barras.

- Condición de carga 1: 1 kip actuando en dirección positiva de las x en los nodos 1, 6, 15, 20, 29, 34, 43, 48, 57, 62 y 71.
- Condición de carga 2: 10 kips actuando en dirección negativa de las y en los nodos 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 15, 16, 17, 18, 19, 20, 24, 71, 72, 73, 74 y 75.
- Condición de carga 3: Condición de carga 1 y 2 actuando en conjunto.
- Los 200 elementos de esta armadura están unidas en 29 grupos (ver tabla C.4).
- Módulo de elasticidad de Young: 30,000 ksi.
- Densidad del peso: 0.283 $\times 10^3$ kips/in^3.
- Máximo esfuerzo permitido para cada miembro (tensión y compresión): 10 ksi.

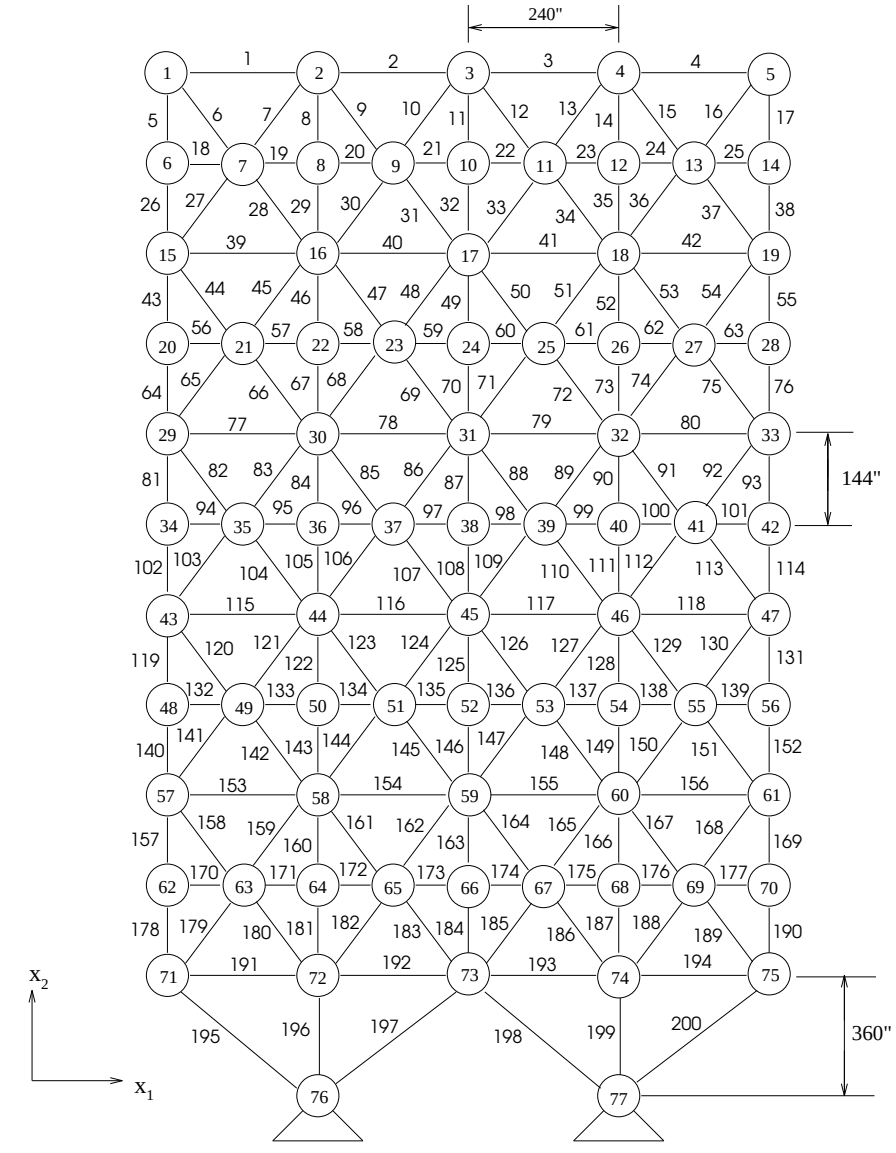


Figura C.3: Armadura plana de 200 barras.

Tabla C.4: Agrupación de miembros para la armadura plana de 200 barras.

Grupo	Miembro
1	1,2,3,4
2	5,8,11,14,17
3	19,20,21,22,23,24
4	18,25,56,63,94,101,132,139,170,177
5	26,29,32,35,38
6	6,7,9,10,12,13,15,16,27,28,30,31,33,34,36,37
7	39,40,41,42
8	43,46,49,52,55
9	57,58,59,60,61,62
10	64,67,70,73,76
11	44,45,47,48,50,51,53,54,65,66,68,69,71,72,74,75
12	77,78,79,80
13	81,84,87,90,93
14	95,96,97,98,99,100
15	102,105,108,111,114
16	82,83,85,86,88,89,91,92,103,104,106,107,109,110,112,113
17	115,116,117,118
18	119,122,125,128,131
19	133,134,135,136,137,138
20	140,143,146,149,152
21	120,121,123,124,126,127,129,130,141,142,144,145,147,148,150,151
22	153,154,155,156
23	157,160,163,166,169
24	171,172,173,174,175,176
25	178,181,184,187,190
26	158,159,161,162,164,165,167,168,179,180,182,183,185,186,188,189
27	191,192,193,194
28	195,197,198,200
29	196,199

Apéndice D

Funciones de diseño ingenieril de piezas

E01: diseño óptimo de una viga soldada

El problema consiste en diseñar una viga soldada que minimice el costo total de fabricación de la pieza, considerando algunas restricciones [148]. La figura D.1 muestra la estructura de la pieza que consiste de la viga A y la soldadura correspondiente para unirla al miembro B. El problema se diseña con cuatro variables: x_1, x_2, x_3, x_4 y restricciones de tensión de corte τ, flexibilidad σ, carga en la barra P_c, y desvío de la viga δ. El objetivo es:
Minimizar: $f(\vec{x}) = 1{,}10471{x_1}^2 x_2 + 0{,}04811 x_3 x_4 (14{,}0 + x_2)$
sujeto a:

$$g_1(\vec{x}) = \tau(\vec{x}) - 13,600 \leq 0$$

$$g_2(\vec{x}) = \sigma(\vec{x}) - 30,000 \leq 0$$

$$g_3(\vec{x}) = x_1 - x_4 \leq 0$$

$$g_4(\vec{x}) = 0{,}10471({x_1}^2) + 0{,}04811 x_3 x_4 (14 + x_2) - 5{,}0 \leq 0$$

$$g_5(\vec{x}) = 0{,}125 - x_1 \leq 0$$

$$g_6(\vec{x}) = \delta(\vec{x}) - 0{,}25 \leq 0$$

$$g_7(\vec{x}) = 6,000 - Pc(\vec{x}) \leq 0$$

con:

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + (2\tau'\tau'')\frac{x_2}{2R} + (\tau'')^2}$$

$$\tau' = \frac{6,000}{\sqrt{2} x_1 x_2}$$

$$\tau'' = \frac{MR}{J}$$

$$M = 6,000 \left(14 + \frac{x_2}{2}\right)$$

$$R = \sqrt{\frac{{x_2}^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}$$

$$J = 2\left\{x_1 x_2 \sqrt{2}\left[\frac{{x_2}^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$$

$$\sigma(\vec{x}) = \frac{504,000}{x_4 {x_3}^2}$$

$$\delta(\vec{x}) = \frac{65,856,000}{(30 \times 10^6) x_4 {x_3}^3}$$

$$Pc(\vec{x}) = \frac{4{,}013(30 \times 10^6)\sqrt{\frac{{x_3}^2 {x_4}^6}{36}}}{196}\left(1 - \frac{x_3\sqrt{\frac{30\times10^6}{4(12\times10^6)}}}{28}\right)$$

con $0{,}1 \leq x_1,\ x_4 \leq 2{,}0$, y $0{,}1 \leq x_2,\ x_3 \leq 10{,}0$.
Mejor solución: $x^* = (0{,}205730, 3{,}470489, 9{,}036624, 0{,}205729)$ con $f(x^*) = 1{,}724852$.

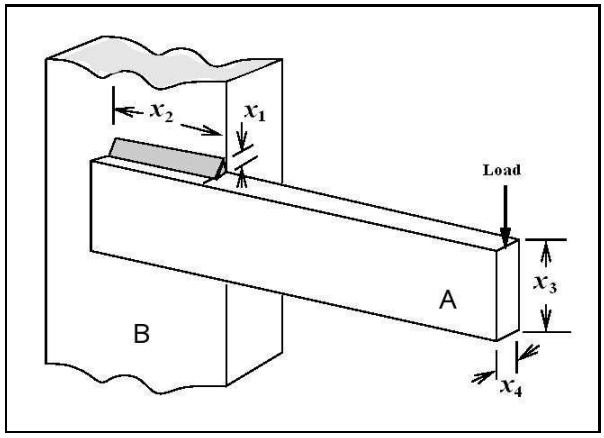


Figura D.1: Viga soldada (Weldem Beam).

E02: diseño óptimo de una válvula de presión

Un tanque que almacena aire comprimido trabaja a una presión de 3,000 psi y a un volumen mínimo de 750 ft^3. Una válvula cilíndrica tapa cada uno de los lados del tanque (ver figura D.2). Utilizando una placa de acero, la cubierta es fabricada en dos partes que luego serán unidas por soldaduras longitudinales al cilindro. El objetivo es minimizar el costo total de fabricación que incluye materiales y soldaduras [163]. Las variables de diseño son: espesor de la tapa x_1, espesor de la cabeza x_2, radio interno x_3, y longitud de la sección cilíndrica de la válvula x_4. Las variables x_1 y x_2 son valores discretos múltiplos de 0.0625 plg. El problema, es entonces:
Minimizar: $f(\vec{x}) = 0{,}6224 x_1 x_3 x_4 + 1{,}7781 x_2 {x_3}^2 + 3{,}1661 {x_1}^2 x_4 + 19{,}84 {x_1}^2 x_3$
sujeto a:

$$g_1(\vec{x}) = -x_1 + 0{,}0193 x_3 \leq 0$$

$$g_2(\vec{x}) = -x_2 + 0{,}00954x_3 \leq 0$$

$$g_3(\vec{x}) = -\pi {x_3}^2 {x_4}^2 - \frac{4}{3}\pi {x_3}^3 + 1{,}296{,}000 \leq 0$$

$$g_4(\vec{x}) = x_4 - 240 \leq 0$$

con $1 \times 0{,}0625 \leq x_1$, $x_2 \leq 99 \times 0{,}0625$, $10{,}0 \leq x_3$, y $x_4 \leq 200{,}0$.
Mejor solución: $x^* = (0{,}8125, 0{,}4375, 42{,}098446, 176{,}636596)$ con $f(x^*) = 6{,}059{,}714335$.

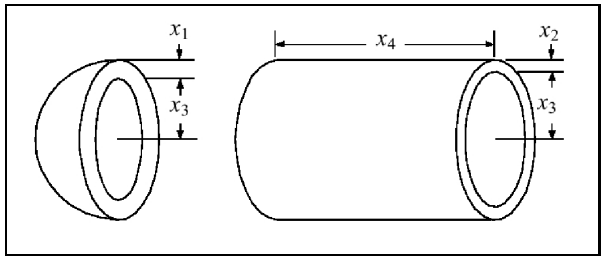


Figura D.2: Válvula de presión (Pressure Vessel).

E03: diseño óptimo de un reductor de velocidad

El diseño de un reductor de velocidad [62] es mostrado en la figura D.3. Se consideran que x_1 es el ancho del rotor, el módulo de potencia es x_2, x_3 es la cantidad de dientes en el piñón, x_4 es la longitud de la primera cavidad, la longitud de la segunda cavidad es x_5, el diámetro de la primera cavidad es x_6, y x_7 el diámetro de la segunda cavidad. Todas las variables son continuas excepto por x_3 que es un valor entero. El peso del reductor de velocidad debe ser minimizado sujeto a restricciones de esfuerzo de curvatura de los dientes del engranaje, superficie de esfuerzo total, flexibilidad transversa de las cavidades y esfuerzo en las cavidades. Formalmente, el problema consiste en:
Minimizar: $f(\vec{x}) = 0{,}7854x_1{x_2}^2(3{,}3333x_3^2 + 14{,}9334x_3 - 43{,}0934) - 1{,}508x_1(x_6^2 + x_7^2) + 7{,}4777(x_6^3 + x_7^3) + 0{,}7854(x_4x_6^2 + x_5x_7^2)$
sujeto a:

$$g_1(\vec{x}) = \frac{27}{x_1x_2^2x_3} - 1 \leq 0$$

$$g_2(\vec{x}) = \frac{397{,}5}{x_1x_2^2x_3^2} - 1 \leq 0$$

$$g_3(\vec{x}) = \frac{1{,}93x_4^3}{x_2x_3x_6^4} - 1 \leq 0$$

$$g_4(\vec{x}) = \frac{1{,}93x_5^3}{x_2x_3x_7^4} - 1 \leq 0$$

$$g_5(\vec{x}) = \frac{1{,}0}{110x_6^3}\sqrt{\left(\frac{745{,}0x_4}{x_2x_3}\right)^2 + 16{,}9 \times 10^6} - 1 \leq 0$$

$$g_6(\vec{x}) = \frac{1{,}0}{85x_7^3}\sqrt{\left(\frac{745{,}0x_5}{x_2x_3}\right)^2 + 157{,}5 \times 10^6} - 1 \leq 0$$

$$g_7(\vec{x}) = \frac{x_2x_3}{40} - 1 \leq 0$$

$$g_8(\vec{x}) = \frac{5x_2}{x_1} - 1 \leq 0$$

$$g_9(\vec{x}) = \frac{x_1}{12x_2} - 1 \leq 0$$

$$g_{10}(\vec{x}) = \frac{1{,}5x_6 + 1{,}9}{x_4} - 1 \leq 0$$

$$g_{11}(\vec{x}) = \frac{1{,}1x_7 + 1{,}9}{x_5} - 1 \leq 0$$

con $2{,}6 \leq x_1 \leq 3{,}6$, $0{,}7 \leq x_2 \leq 0{,}8$, $17 \leq x_3 \leq 28$, $7{,}3 \leq x_4 \leq 8{,}3$, $7{,}8 \leq x_5 \leq 8{,}3$, $2{,}9 \leq x_6 \leq 3{,}9$, y $5{,}0 \leq x_7 \leq 5{,}5$.
Mejor solución: $x^* = (3{,}500000, 0{,}7, 17, 7{,}300000, 7{,}800000, 3{,}350214, 5{,}286683)$ donde $f(x^*) = 2,996{,}348165$.

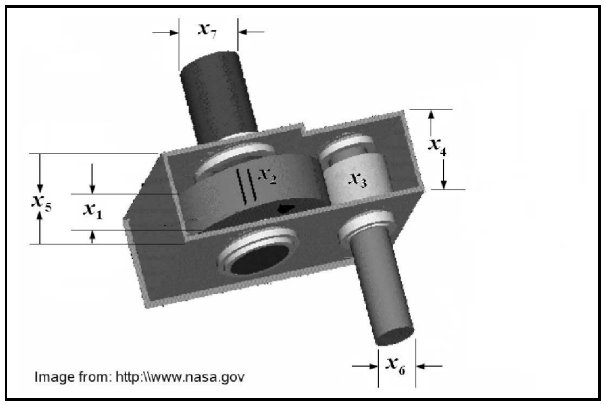


Figura D.3: Reductor de velocidad (Speed Reducer)

E04: diseño óptimo para la tensión/compresión de un resorte

Este problema [5] [10] busca minimizar el peso de la tensión/compresión de un resorte (figura D.4), sujeto a restricciones de mínima flexibilidad, estiramiento, frecuencia de sobrecarga y límites del diámetro externo. El problema es modelado utilizando tres variables de diseño: el diámetro del alambre x_1, el diámetro medio de cada onda (rulo) del resorte x_2, y el número de ondas activas en el resorte x_3. La formulación matemática del problema es:
Minimizar: $f(\vec{x}) = (x_3 + 2)x_2x_1^2$
sujeto a:

$$g_1(\vec{x}) = 1 - \frac{x_2^3x_3}{7,178x_1^4} \leq 0$$

$$g_2(\vec{x}) = \frac{4x_2^2 - x_1x_2}{12,566(x_2x_1^3) - x_1^4} + \frac{1}{5,108x_1^2} - 1 \leq 0$$

$$g_3(\vec{x}) = 1 - \frac{140{,}45x_1}{x_2^2x_3} \leq 0$$

$$g_4(\vec{x}) = \frac{x_2 + x_1}{1{,}5} - 1 \leq 0$$

con $0{,}05 \leq x_1 \leq 2{,}0$, $0{,}25 \leq x_2 \leq 1{,}3$, y $2{,}0 \leq x_3 \leq 15{,}0$.
Mejor solución: $x^* = (0{,}051690, 0{,}356750, 11{,}287126)$ con $f(x^*) = 0{,}012665$.

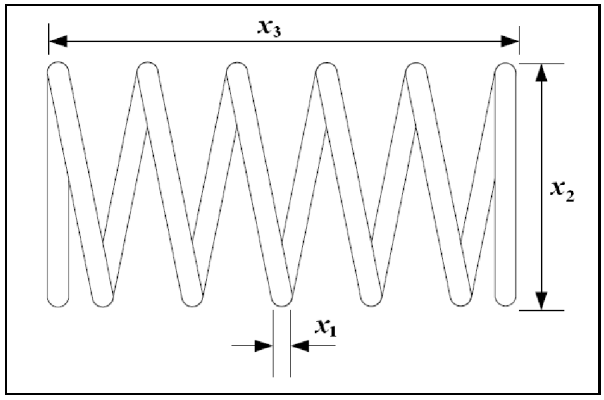


Figura D.4: Tensión/compresión de un resorte (Tension/Compression Spring).

Apéndice E

Problema de despacho de cargas eléctricas

En sistemas operativos de potencia, el problema de planificar unidades de generación para satisfacer una demanda de electricidad considerando restricciones, es denominado *Unit Commitment* [197].
Un subproblema del *Unit Commitment* es el de despacho de energía (*Economic Dispatch*, abreviado como ED), en el cual la planificación óptima de generación de energía en un cierto periodo de tiempo, debe ser determinado. Esa planificación debe minimizar el costo total de producción mientras satisface restricciones de igualdad (balance de potencia) y restricciones de desigualdad (límites operacionales). Por ende, la formulación matemática del problema de ED es [197]:

$$\text{minimizar TC} = \sum_{i=1}^{N} F_i(P_i) \tag{E.1}$$

sujeto a:

$$\sum_{i=1}^{N} P_i = P_D \qquad \text{(Restricción de Balance de Potencia)} \tag{E.2}$$

y

$$Pmin_i \leq P_i \leq Pmax_i \qquad \text{(Restricción de Límite Operacional)} \tag{E.3}$$

donde TC es el costo total de producción, F_i es el costo de combustible requerido por el generador i, P_i la potencia producida por el generador i, N la cantidad de generadores, P_D la demanda de potencia requerida, $Pmin_i$ y $Pmax_i$ las potencias de salida mínimas y máximas del generador i.
Dependiendo de las características de la función de costo de combustible F, puede ser categorizada como suave (*smooth*) o no suave (*non-smooth*). Esta característica define dos tipos de problemas:

a. **ED con función de costo suave**

Es el más simple de los dos tipos y la función de costo puede ser representada como una función cuadrática simple [197]:

$$F_i(P_i) = a_i P_i^2 + b_i P_i + c_i \tag{E.4}$$

con a_i, b_i y c_i los coeficientes del costo del consumo de combustible del generador i.

b. **ED con función de costo no suave**

Se pueden observar algunos efectos *valve-points*, por lo tanto la función objetivo incluye múltiples puntos no diferenciables. Esta característica dificulta el problema, y de hecho, la tarea de descubrir un mínimo global para esta función objetivo permanece sin solución [172].
Comparada con la función de costo suave, a ésta se le agregan funciones sinusoidales [194]:

$$F_i(P_i) = a_i P_i^2 + b_i P_i + c_i + |e_i sin(f_i(Pmin_i - P_i))| \tag{E.5}$$

con e_i y f_i los coeficientes de costo de combustible de la unidad i con efectos *valve-point*.

Datos para el problema de despacho de cargas

En la tabla E.1 se muestran los valores para las tres unidades: potencias mínimas y máximas, y coeficientes de los costos (a, b y c), correspondientes al Caso A (función suave). Los datos en la tabla E.2 son los mismos que para el Caso A pero se agregan los valores de los coeficientes para las funciones no suaves, es decir e y f, resultando en el Caso B. Los datos para el Caso C se encuentran en la tabla E.3.

Tabla E.1: Datos para **Caso A** - 3 unidades generadoras con una función suave [197].

Unidad	Pmin(MW)	Pmax(MW)	a	b	c
1	150	600	0.001562	7.92	561
2	100	400	0.001940	7.85	310
3	50	200	0.004820	7.97	78

Tabla E.2: Datos para **Caso B** - 3 unidades generadoras con una función no suave [194].

Unidad	Pmin(MW)	Pmax(MW)	a	b	c	e	f
1	100	600	0.001562	7.92	561	300	0.0315
2	100	400	0.001940	7.85	310	200	0.042
3	50	200	0.004820	7.97	78	150	0.063

Tabla E.3: Datos para **Caso C** - 40 unidades generadoras con una función no suave [172].

Unidad	Pmin(MW)	Pmax(MW)	a	b	c	e	f
1	36	114	0.00690	6.73	94.705	100	0.084
2	36	114	0.00690	6.73	94.705	100	0.084
3	60	120	0.02028	7.07	309.54	100	0.084
4	80	190	0.00942	8.18	369.03	150	0.063
5	47	97	0.0114	5.35	148.89	120	0.077
6	68	140	0.01142	8.05	222.33	100	0.084
7	110	300	0.00357	8.03	287.71	200	0.042
8	135	300	0.00492	6.99	391.98	200	0.042
9	135	300	0.00573	6.60	455.76	200	0.042
10	130	300	0.00605	12.9	722.82	200	0.042
11	94	375	0.00515	12.9	635.20	200	0.042
12	94	375	0.00569	12.8	654.69	200	0.042
13	125	500	0.00421	12.5	913.40	300	0.035
14	125	500	0.00752	8.84	1760.40	300	0.035
15	125	500	0.00708	9.15	1728.30	300	0.035
16	125	500	0.00708	9.15	1728.30	300	0.035
17	220	500	0.00313	7.97	647.85	300	0.035
18	220	500	0.00313	7.95	649.69	300	0.035
19	242	550	0.00313	7.97	647.83	300	0.035
20	242	550	0.00313	7.97	647.81	300	0.035
21	254	550	0.00298	6.63	785.96	300	0.035
22	254	550	0.00298	6.63	785.96	300	0.035
23	254	550	0.00284	6.66	794.53	300	0.035
24	254	550	0.00284	6.66	794.53	300	0.035
25	254	550	0.00277	7.10	801.32	300	0.035
26	254	550	0.00277	7.10	801.32	300	0.035
27	10	150	0.52124	3.33	1055.10	120	0.077
28	10	150	0.52124	3.33	1055.10	120	0.077
29	10	150	0.52124	3.33	1055.10	120	0.077
30	47	97	0.01140	5.35	148.89	120	0.077
31	60	190	0.00160	6.43	222.92	150	0.063
32	60	190	0.00160	6.43	222.92	150	0.063
33	60	190	0.00160	6.43	222.92	150	0.063
34	90	200	0.0001	8.95	107.87	200	0.042
35	90	200	0.0001	8.62	116.58	200	0.042
36	90	200	0.0001	8.62	116.58	200	0.042
37	25	110	0.0161	5.88	307.45	80	0.098
38	25	110	0.0161	5.88	307.45	80	0.098
39	25	110	0.0161	5.88	307.45	80	0.098
40	242	550	0.00313	7.97	647.83	300	0.035

Apéndice F

Funciones multiobjetivo

La serie de problemas no escalable que se describen a continuación fue propuesta por Okabe [132]. Cada problema posee 2 y 3 variables respectivamente con sólo 2 funciones objetivo de minimización cada uno.

OKA1:

$$f_1(\vec{x}) = cos\left(\frac{\pi}{12}\right)x_1 - sin\left(\frac{\pi}{12}\right)x_2$$

$$f_2(x) = \sqrt{2\pi} - \sqrt{\left|cos\left(\frac{\pi}{12}\right)x_1 - sin\left(\frac{\pi}{12}\right)x_2\right|}$$

$$+2\left|sin\left(\frac{\pi}{12}\right)x_1 + cos\left(\frac{\pi}{12}\right)x_2 - 3cos\left(cos\left(\frac{\pi}{12}\right)x_1 - sin\left(\frac{\pi}{12}\right)x_2\right) - 3\right|^{\frac{1}{3}}$$

con $6sin(\frac{\pi}{12}) \leq x_1 \leq 6sin(\frac{\pi}{12}) + 2\pi cos(\frac{\pi}{12})$ y $-2\pi sin(\frac{\pi}{12}) \leq x_2 \leq 6cos(\frac{\pi}{12})$.

OKA2:

$$f_1(\vec{x}) = x_1$$

$$f_2(x) = 1 - \frac{1}{4\pi^2}(x_1 + \pi)^2 + |x_2 - 5cos(x_1)|^{\frac{1}{3}} + |x_3 - 5sin(x_1)|^{\frac{1}{3}}$$

con $-\pi \leq x_1 \leq \pi$ y $-5 \leq x_2, x_3 \leq 5$.

Las siguientes cinco funciones forman parte de una serie de funciones multiobjetivo de minimización [76]. Los autores de la serie propusieron un conjunto de transformaciones que se aplican secuencialmente a las variables de decisión. Cada transformación incorpora una característica particular a cada problema. Todos los problemas utilizan el siguiente formato:

$$\text{Dado } \vec{z} = [z_1, \ldots, z_k, z_{k+1}, \ldots, z_n]$$

$$\text{Minimizar } f_{m=1:M}(\vec{x}) = Dx_M + S_m h_m(x_1, \ldots, x_{M-1})$$

$$\text{donde } \vec{x} = [x_1, \ldots, x_M]$$
$$= [\max(t_M^p, A_1)(t_1^p - 0{,}5) + 0{,}5, \ldots, \max(t_M^p, A_{M-1})(t_{M-1}^p - 0{,}5) + 0{,}5, t_M^p]$$
$$\vec{t}^p = [t_1^p, \ldots, t_M^p] \leftarrow\!\shortmid\, \vec{t}^{p-1} \leftarrow\!\shortmid\, \ldots \vec{t}^1 \leftarrow\!\shortmid\, \vec{z}_{[0,1]}$$
$$\vec{z}_{[0,1]} = [z_{1,[0,1]}, \ldots, z_{n,[0,1]}]$$
$$= [z_1/z_{1,\max}, \ldots, z_n/z_{n,\max}]$$

siendo $\vec{z}$ el vector de variables de decisión con $0 \leq z_i \leq z_{i,\max}$, D, $A_{1:M-1}$ y $S_{1:M}$ constantes que modifican la posición y escalan el frente de Pareto.
Las funciones $h_{1:M}$ definen la forma del frente de Pareto el cual puede ser lineal, convexo, cóncavo o una combinación de convexo y cóncavo (mixto). Tales características pueden ser obtenidas mediante el uso de las siguientes funciones:

$$\text{linear}_1(x_1, \ldots, x_{M-1}) = \prod_{i=1}^{M-1} x_i$$

$$\text{linear}_{m=2:M-1}(x_1, \ldots, x_{M-1}) = \left(\prod_{i=1}^{M-m} x_i \right) (1 - x_{M-m+1})$$
$$\text{linear}_M(x_1, \ldots, x_{M-1}) = 1 - x_1$$

$$\text{convex}_1(x_1, \ldots, x_{M-1}) = \prod_{i=1}^{M-1} (1 - cos(x_i \pi/2))$$
$$\text{convex}_{m=2:M-1}(x_1, \ldots, x_{M-1}) = \left(\prod_{i=1}^{M-m} (1 - cos(x_i \pi/2)) \right) (1 - sin(x_{M-m+1} \pi/2))$$
$$\text{convex}_M(x_1, \ldots, x_{M-1}) = 1 - sin(x_1 \pi/2)$$

$$\text{concave}_1(x_1, \ldots, x_{M-1}) = \prod_{i=1}^{M-1} sin(x_i \pi/2)$$
$$\text{concave}_{m=2:M-1}(x_1, \ldots, x_{M-1}) = \left(\prod_{i=1}^{M-m} sin(x_i \pi/2) \right) cos(x_{M-m+1} \pi/2)$$
$$\text{concave}_M(x_1, \ldots, x_{M-1}) = cos(x_1 \pi/2)$$

$$\text{mixed}_M(x_1, \ldots, x_{M-1}) = \left(1 - x_1 - \frac{cos(2A\pi x_1 + \pi/2)}{2A\pi} \right)^{\alpha}$$

Para obtener un frente de Pareto mixto se utiliza: $\alpha > 1$ para que sea cóncavo, $\alpha < 1$ para que sea convexo y con $\alpha = 1$ será lineal. El número de segmentos cóncavos-convexos es A.
Otras características son incorporadas a través de un conjunto de transformaciones. Los autores de la serie distinguen tres tipos de transformaciones basados en lo que ellos consideran importante a la hora de diseñar un problema multiobjetivo. Las transformaciones de sesgo (*bias*) producen un sesgo en la zona de soluciones (*fitness landscape*) el cual puede ser polinomial, planar o algún otro tipo que depende exclusivamente de las variables que se utilicen. Las transformaciones de cambio (*shift*) mueven la ubicación de los valores óptimos y son empleadas para aplicar cambios lineales o producir problemas deceptivos (poseen más de dos óptimos y las características del paisaje propician a que se encuentre el óptimo falso en vez del verdadero) y multimodales. Las transformaciones de reducción son usadas para producir problemas no separables ya que combinan los valores de varias variables en una única. El conjunto de transformaciones es el siguiente:

$$\text{b\_poly}(y, \alpha) = y^{\alpha}$$

$$\text{b\_flat}(y, A, B, C) = A + \min(0, \lfloor y - B \rfloor)\frac{A(B-y)}{B} - \min(0, \lfloor C - y \rfloor)\frac{(1-A)(y-C)}{1-C}$$

$$\text{b\_param}(y, u(\vec{y'}, A, B, C) = y^{B+(C-B)(A-(1-2u(\vec{y'}))|\lfloor 0{,}5-u(\vec{y'})\rfloor+A|)}$$

$$\text{s\_linear}(y, A) = \frac{|y-A|}{|\lfloor A-y \rfloor + A|}$$

$$\text{s\_decept}(y, A, B, C) =$$

$$1 + (|y-A| - B)\left(\frac{\lfloor A-y+B \rfloor(1-C+\frac{A-B}{B})}{A-B} + \frac{\lfloor A+B-y \rfloor(1-C+\frac{1-A-B}{B})}{1-A-B} + \frac{1}{B}\right)$$

$$\text{s\_multi}(y, A, B, C) =$$

$$\left(1 + cos\left((4A+2)\pi\left(0{,}5 - \frac{|y-C|}{2(\lfloor C-y \rfloor + C)}\right)\right) + 4B\left(\frac{|y-C|}{2(\lfloor C-y \rfloor + C)}\right)^2\right)/(b+2)$$

$$\text{r\_sum}(\vec{y}, \vec{w}) = \frac{\sum_{i=1}^{|y|} w_1 y_i}{\sum_{i=1}^{|\vec{y}|} w_i}$$

$$\text{r\_nonsep}(\vec{y}, A) = \frac{\sum_{j=1}^{|y|}\left(y_j + \sum_{k=0}^{A-2}\left|y_j - y_{1+(j+k)mod|y|}\right|\right)}{\frac{|\vec{y}|}{A}\lceil\frac{A}{2}\rceil(1+2A-2\lceil\frac{A}{2}\rceil)}$$

Luego, las siguientes funciones multiobjetivo utilizan las transformaciones anteriormente descriptas:

WFG1:

$$f_{m=1:M-1}(\vec{x}) = x_M + S_m \text{convex}_m(x_1, \ldots, x_{M-1})$$

$$f_M(\vec{x}) = x_M + S_M \text{mixed}_M(x_1, \ldots, x_{M-1})$$

donde:

$$y_{i=1:M-1} = \text{r\_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [2((i-1)k/(M-1)+1), \ldots, 2ik/(M-1)])$$

$$y_M = \text{r\_sum}([y'_{k+1}, \ldots, y'_n], [2(k+1), \ldots, 2n])$$

$$y'_{i=1:n} = \text{b\_poly}(y''_i, 0{,}02)$$

$$y''_{i=1:k} = y'''_i$$

$$y''_{i=k+1:n} = \text{b\_flat}(y'''_i, 0{,}8, 0{,}75, 0{,}85)$$

$$y'''_{i=1:k} = z_{i,[0,1]}$$

$$y'''_{i=k+1:n} = \text{s\_linear}(z_{i,[0,1]}, 0{,}35)$$

WFG3:

$$f_{m=1:M}(\vec{x}) = x_M + S_m \text{linear}_m(x_1, \ldots, x_{M-1})$$

donde:

$$y_{i=1:M-1} = \text{r\_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1])$$

$$y_M = \text{r\_sum}([y'_{k+1}, \ldots, y'_{k+l/2}], [1, \ldots, 1])$$

$$y'_{i=1:k} = y''_i$$

$$y'_{i=k+1:k+l/2} = \text{r\_nonsep}(y''_{k+2(i-k)-1}, y''_{k+2(i-k)}], 2)$$

$$y''_{i=1:k} = z_{i,[0,1]}$$

$$y''_{i=k+1:n} = \text{s\_linear}(z_{i,[0,1]}, 0{,}35)$$

WFG5:

$$f_{m=1:M}(\vec{x}) = x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

donde:

$$y_{i=1:M-1} = \text{r\_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1])$$

$$y_M = \text{r\_sum}([y'_{k+1}, \ldots, y'_n], [1, \ldots, 1])$$

$$y''_{i=1:n} = \text{s\_decept}(z_{i,[0,1]}, 0{,}35, 0{,}001, 0{,}05)$$

WFG7:

$$f_{m=1:M}(\vec{x}) = x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

donde:

$$y_{i=1:M-1} = \text{r\_sum}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], [1, \ldots, 1])$$

$$y_M = \text{r\_sum}([y'_{k+1}, \ldots, y'_n], [1, \ldots, 1])$$

$$y'_{i=1:k} = y''_i$$

$$y'_{i=k+1:n} = \text{s\_linear}(y'', 0{,}35)$$

$$y''_{i=1:k} = \text{b\_param}(z_{i,[0,1]}, \text{r\_sum}([z_{i+1,[0,1]}, \ldots, z_{n,[0,1]}], [1, \ldots, 1]), 0{,}98/49{,}98, 0{,}02, 50)$$
$$y''_{i=k+1:n} = z_{i,[0,1]}$$

WFG9:

$$f_{m=1:M}(\vec{x}) = x_M + S_m \text{concave}_m(x_1, \ldots, x_{M-1})$$

donde:

$$y_{i=1:M-1} = \text{r\_nonsep}([y'_{(i-1)k/(M-1)+1}, \ldots, y'_{ik/(M-1)}], k/(M-1))$$
$$y_M = \text{r\_nonsep}([y'_{k+1}, \ldots, y'_n], l)$$
$$y'_{i=1:k} = \text{s\_decept}(y''_i, 0{,}35, 0{,}001, 0{,}05)$$
$$y'_{i=k+1:n} = \text{s\_multi}(y''_i, 30, 95, 0{,}35)$$
$$y''_{i=1:n-1} = \text{b\_param}(z_{i,[0,1]}, \text{r\_sum}([z_{i+1,[0,1]}, \ldots, z_{n,[0,1]}], [1, \ldots, 1]), 0{,}98/49{,}98, 0{,}02, 50)$$
$$y''_n = z_{n,[0,1]}$$

Los autores de esta serie sugieren utilizar estos problemas con 24 variables ($k = 4$ y $n = 24$), con las siguientes constantes:

$$z_{i=1:n,max} = 2i$$
$$S_{m=1:M} = 2m$$
$$A_1 = 1$$
$$D = 1$$
$$A_{2:M-1} = \begin{cases} 0, & \text{para WFG3} \\ 1, & \text{en otro} \end{cases}$$

Apéndice G

Resultados de LHD para las 20 configuraciones

G.1. Funciones sin restricciones

Tabla G.1: Resultados de LHD para f1.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.0000	0.0000
C2	0.0000	0.0000	0.0000
C3	0.0000	0.0000	0.0000
C4	0.0000	0.0000	0.0000
C5	0.0120	495.7381	520.7323
C6	0.0000	59.0030	265.6728
C7	0.0000	0.0000	0.0000
C8	0.0000	567.5149	502.2964
C9	0.0000	0.0000	0.0000
C10	0.0000	0.0011	0.0019
C11	0.0000	0.0000	0.0000
C12	0.0000	0.0000	0.0000
C13	0.0000	0.0000	0.0000
C14	0.0000	0.0000	0.0000
C15	0.0000	0.0000	0.0000
C16	0.0000	0.0000	0.0000
C17	0.0000	10.1125	51.9600
C18	0.0000	0.0000	0.0000
C19	0.0000	0.0000	0.0000
C20	0.7305	3.3569	1.9965

Tabla G.2: Resultados de LHD para f2.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.0000	0.0000
C2	0.0000	0.0000	0.0000
C3	0.0000	0.0000	0.0000
C4	0.0000	0.0000	0.0000
C5	0.0040	2.8195	2.0856
C6	0.0000	0.4703	1.1459
C7	0.0000	0.0000	0.0000
C8	0.0000	3.3916	2.3717
C9	0.0000	0.0000	0.0000
C10	0.0000	0.0009	0.0006
C11	0.0000	0.0000	0.0000
C12	0.0000	0.0000	0.0000
C13	0.0000	0.0000	0.0000
C14	0.0000	0.0000	0.0000
C15	0.0000	0.0000	0.0000
C16	0.0000	0.0000	0.0000
C17	0.0000	0.0492	0.2040
C18	0.0000	0.0000	0.0000
C19	0.0000	0.0000	0.0000
C20	0.0918	0.4488	0.1588

Tabla G.3: Resultados de LHD para f3.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0568	22.2483	65.6921
C2	0.0002	4.0322	15.8772
C3	0.0026	0.1122	0.1764
C4	0.0388	79.5478	303.3347
C5	2,743.4170	7,406.8070	1,722.5367
C6	0.0001	656.3435	704.7380
C7	1.7333	280.4966	633.7163
C8	0.3053	5,632.7448	2,079.7323
C9	0.0050	0.6031	2.7304
C10	6,343.0828	10,446.9819	1,941.9414
C11	0.5910	43.4139	95.9448
C12	382.7064	6,171.6045	3,477.3539
C13	751.6715	4,344.3146	2,993.6866
C14	0.0433	4.6882	12.6191
C15	153.8613	3,034.5542	2,280.0426
C16	0.0410	2.1362	5.6032
C17	0.0339	96.3450	195.2295
C18	0.0017	6.3830	32.9937
C19	0.0019	0.0352	0.0447
C20	1,735.5601	4,157.1381	1,040.1389

Tabla G.4: Resultados de LHD para f4.

Configuración	Mejor	Media	Desv. Estándar
C1	1.1642	3.7351	2.2627
C2	0.2217	1.3169	0.9880
C3	0.4775	1.9624	0.7673
C4	0.6372	2.3277	1.1287
C5	13.7672	25.0643	2.6854
C6	0.3042	4.7466	1.8731
C7	0.3767	2.0233	1.1301
C8	1.0491	19.6559	2.2552
C9	0.2293	2.1845	1.4527
C10	8.0031	11.5213	0.9049
C11	0.2271	0.9778	0.5571
C12	0.3183	2.3924	1.2979
C13	3.1452	11.6338	7.1397
C14	0.4013	2.3516	1.4762
C15	0.1753	2.8076	3.5082
C16	0.4698	2.1437	0.8569
C17	0.2301	1.9170	1.0044
C18	0.1808	1.2664	0.7279
C19	0.0917	0.8568	0.4233
C20	5.6336	8.0081	0.8820

Tabla G.5: Resultados de LHD para f5.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0561	93.5287	123.8890
C2	0.0065	76.9872	130.8003
C3	0.0369	28.6612	29.3133
C4	0.0044	33.6414	29.6005
C5	213.1870	991.2957	1,669.3940
C6	0.1424	2,403.1877	8,368.2212
C7	0.2545	35.0599	40.0365
C8	4.5772	384.7391	3,716.6911
C9	0.0692	25.4756	28.1595
C10	108.1180	269.8151	111.1116
C11	0.0880	34.8072	31.4042
C12	0.1137	38.8705	27.2764
C13	0.2497	204.7050	393.0565
C14	0.0011	32.6562	35.1220
C15	0.0504	38.9621	36.8329
C16	0.0190	25.7134	27.5064
C17	0.0000	61.1255	86.8515
C18	0.1617	25.6727	28.9454
C19	0.0485	22.6960	22.4535
C20	338.1376	750.5860	393.1341

Tabla G.6: Resultados de LHD para f6.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.2167	0.7493
C2	0.0000	0.0667	0.2494
C3	0.0000	1.0833	1.6886
C4	0.0000	0.1000	0.2380
C5	8.0000	848.8333	556.7777
C6	1.0000	1,542.3333	1,357.3723
C7	0.0000	4.8500	12.0590
C8	4.0000	1,441.7833	822.0549
C9	0.0000	26.7333	69.7821
C10	0.0000	5.0667	20.3379
C11	0.0000	8.3500	29.6018
C12	0.0000	7.0167	17.3659
C13	0.0000	2.9667	7.2960
C14	0.0000	0.0167	0.0898
C15	0.0000	57.5000	175.3141
C16	0.0000	1.1167	1.5258
C17	0.0000	163.8333	250.4216
C18	0.0000	1.8500	8.7884
C19	0.0000	0.0000	0.0000
C20	1.0000	8.1000	2.9166

Tabla G.7: Resultados de LHD para f7.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.0000	0.0000
C2	0.0000	0.0000	0.0000
C3	0.0000	0.0000	0.0000
C4	0.0000	0.0000	0.0000
C5	0.0000	0.0255	0.0520
C6	0.0000	0.0276	0.1487
C7	0.0000	0.0000	0.0000
C8	0.0000	0.0888	0.1239
C9	0.0000	0.0000	0.0000
C10	0.0000	0.0000	0.0000
C11	0.0000	0.0000	0.0000
C12	0.0000	0.0000	0.0000
C13	0.0000	0.0000	0.0000
C14	0.0000	0.0000	0.0000
C15	0.0000	0.0000	0.0000
C16	0.0000	0.0000	0.0000
C17	0.0000	0.0000	0.0000
C18	0.0000	0.0000	0.0000
C19	0.0000	0.0000	0.0000
C20	0.0000	0.0000	0.0000

Tabla G.8: Resultados de LHD para f8.

Configuración	Mejor	Media	Desv. Estándar
C1	-12,569.4866	-12,504.6769	139.8579
C2	-11,641.7169	-10,675.9607	360.7024
C3	-12,569.4866	-12,528.3776	108.6277
C4	-11,523.2786	-10,455.1747	451.4855
C5	-11,700.9222	-10,473.6748	517.0456
C6	-11,128.4706	-9,436.7718	408.1703
C7	-12,569.4866	-12,318.5022	175.0176
C8	-11,009.4370	-9,850.5386	466.9958
C9	-12,095.7333	-10,322.8353	558.9253
C10	-12,569.4866	-12,046.4465	194.4314
C11	-11,996.8175	-10,387.5116	630.3974
C12	-12,450.8893	-11,829.8374	278.0158
C13	-10,674.4542	-9,461.2632	554.2694
C14	-12,214.1716	-10,465.0646	506.1814
C15	-12,451.0483	-11,343.0086	774.3032
C16	-12,569.4866	-12,544.2012	61.1695
C17	-12,095.7332	-11,060.9589	328.2468
C18	-11,641.7169	-10,146.2855	636.8673
C19	-12,569.4866	-12,564.8970	24.7160
C20	-12,450.0481	-11,143.0080	777.3320

Tabla G.9: Resultados de LHD para f9.

Configuración	Mejor	Media	Desv. Estándar
C1	1.9899	12.2544	5.9961
C2	14.9244	49.9399	13.0801
C3	1.9899	7.0486	3.1189
C4	13.9294	49.3147	11.1338
C5	19.9068	60.1307	12.0696
C6	21.8891	68.6590	15.4598
C7	5.9732	14.6255	4.0859
C8	25.8689	67.6118	14.8536
C9	15.9193	53.4679	12.5393
C10	2.1060	12.3270	4.0647
C11	13.9294	51.4713	16.3192
C12	23.5141	33.1849	4.7362
C13	30.8437	68.9657	20.8651
C14	14.9244	45.7245	13.9546
C15	7.9597	33.6701	19.7036
C16	4.9751	13.1040	6.8153
C17	18.9043	44.2091	7.1505
C18	20.8941	52.6533	11.6570
C19	0.0000	2.6213	1.8611
C20	0.4903	2.8268	2.0872

Tabla G.10: Resultados de LHD para f10.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.3593	0.6003
C2	0.0000	0.7646	0.9076
C3	0.0000	0.9711	0.5930
C4	0.0000	0.0000	0.0000
C5	4.8133	8.4884	1.5068
C6	0.0000	8.6142	2.1742
C7	0.0000	1.7945	0.6044
C8	4.8117	10.0557	1.6480
C9	0.0000	1.9608	1.3262
C10	0.0007	0.3725	0.4088
C11	0.0000	0.7821	0.7809
C12	0.0000	0.4538	0.7902
C13	0.0000	2.5981	1.4476
C14	0.0000	0.0000	0.0000
C15	0.0000	1.4646	1.2643
C16	0.0000	0.0000	0.0000
C17	0.0000	4.7447	1.2411
C18	0.0000	0.1039	0.3930
C19	0.0000	0.0000	0.0000
C20	0.0857	1.0846	0.4742

Tabla G.11: Resultados de LHD para f11.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.0483	0.0465
C2	0.0000	0.0272	0.0329
C3	0.0000	0.0241	0.0252
C4	0.0000	0.0104	0.0194
C5	0.0729	3.9004	4.4764
C6	0.0000	0.8523	1.5268
C7	0.0000	0.0265	0.0390
C8	0.0004	7.2335	6.0837
C9	0.0000	0.0571	0.1037
C10	0.0000	0.1222	0.0690
C11	0.0000	0.0245	0.0207
C12	0.0000	0.0111	0.0113
C13	0.0000	0.0819	0.1600
C14	0.0000	0.0101	0.0180
C15	0.0000	0.0320	0.0467
C16	0.0000	0.0232	0.0358
C17	0.0000	0.2364	0.3627
C18	0.0000	0.0171	0.0259
C19	0.0000	0.0087	0.0125
C20	0.6332	0.9994	0.0705

Tabla G.12: Resultados de LHD para f12.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.0000	0.0000
C2	0.0000	0.0000	0.0000
C3	0.0000	0.0000	0.0000
C4	0.0000	0.0069	0.0259
C5	0.0000	0.0078	0.0155
C6	0.0000	0.0057	0.0154
C7	0.0000	0.0000	0.0000
C8	0.0000	0.0218	0.0291
C9	0.0000	0.0017	0.0093
C10	0.0000	0.0017	0.0093
C11	0.0000	0.0052	0.0156
C12	0.0000	0.0035	0.0129
C13	0.0000	0.0035	0.0186
C14	0.0000	0.0035	0.0186
C15	0.0000	0.0121	0.0257
C16	0.0000	0.0000	0.0000
C17	0.0000	0.0001	0.0004
C18	0.0000	0.0000	0.0000
C19	0.0000	0.0000	0.0000
C20	0.0000	0.0001	0.0001

Tabla G.13: Resultados de LHD para f13.

Configuración	Mejor	Media	Desv. Estándar
C1	0.0000	0.0007	0.0027
C2	0.0000	0.0026	0.0046
C3	0.0000	0.0024	0.0037
C4	0.0000	0.0029	0.0102
C5	0.0000	0.4837	0.4707
C6	0.0000	0.1809	0.1971
C7	0.0000	0.0044	0.0093
C8	0.0000	1.1590	0.7241
C9	0.0000	0.0169	0.0377
C10	0.0000	0.0018	0.0046
C11	0.0000	0.0214	0.0449
C12	0.0000	0.0539	0.1894
C13	0.0000	0.1161	0.3403
C14	0.0000	0.0062	0.0132
C15	0.0000	0.0330	0.1436
C16	0.0000	0.0029	0.0059
C17	0.0000	0.0155	0.0217
C18	0.0000	0.0029	0.0049
C19	0.0000	0.0004	0.0020
C20	0.0001	0.0042	0.0049

G.2. Funciones con restricciones

Tabla G.14: Resultados de LHD para g1.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-13.7110	-14.4780	0.6477	SI
C2	-15.0001	-15.0001	0.0000	SI
C3	-13.7143	-14.3374	0.7469	SI
C4	-15.0001	-15.0001	0.0000	SI
C5	-15.0001	-15.0001	0.0000	SI
C6	-15.0001	-15.0001	0.0000	SI
C7	-15.0001	-15.0001	0.0000	SI
C8	-15.0001	-15.0001	0.0000	SI
C9	-14.7369	-14.5095	0.5211	SI
C10	-15.0001	-14.9926	0.0238	SI
C11	-15.0001	-15.0001	0.0000	SI
C12	-15.0001	-15.0001	0.0000	SI
C13	-15.0001	-15.0001	0.0000	SI
C14	-14.9360	-14.4334	0.7263	SI
C15	-15.0001	-15.0001	0.0001	SI
C16	-15.0001	-15.0001	0.0000	SI
C17	-15.0001	-15.0001	0.0000	SI
C18	-15.0001	-15.0001	0.0000	SI
C19	-15.0001	-15.0001	0.0000	SI
C20	-15.0001	-15.0001	0.0001	SI

Tabla G.15: Resultados de LHD para g2.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-0.8036	-0.7780	0.0196	SI
C2	-0.8036	-0.7842	0.0161	SI
C3	-0.8036	-0.7865	0.0136	SI
C4	-0.8036	-0.7969	0.0070	SI
C5	-0.8035	-0.7945	0.0058	SI
C6	-0.8036	-0.7957	0.0072	SI
C7	-0.8036	-0.7797	0.0152	SI
C8	-0.7808	-0.6936	0.0575	SI
C9	-0.8036	-0.7943	0.0096	SI
C10	-0.7867	-0.7657	0.0139	SI
C11	-0.8036	-0.7956	0.0055	SI
C12	-0.8036	-0.7722	0.0347	SI
C13	-0.7946	-0.7354	0.0444	SI
C14	-0.8036	-0.7956	0.0063	SI
C15	-0.8034	-0.7926	0.0078	SI
C16	-0.8036	-0.7890	0.0143	SI
C17	-0.8035	-0.7941	0.0103	SI
C18	-0.8036	-0.7671	0.0281	SI
C19	-0.8035	-0.7961	0.0060	SI
C20	-0.8036	-0.7924	0.0075	SI

Tabla G.16: Resultados de LHD para g3.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-1.0000	-1.0001	0.0004	SI
C2	-1.0010	-1.0010	0.0000	SI
C3	-1.0001	-1.0003	0.0005	SI
C4	-1.0010	-1.0009	0.0000	SI
C5	-1.0010	-1.0005	0.0011	SI
C6	-1.0010	-1.0006	0.0013	SI
C7	-1.0010	-1.0010	0.0000	SI
C8	-1.0010	-1.0010	0.0000	SI
C9	-1.0001	-1.0003	0.0006	SI
C10	-0.9941	-0.9487	0.0282	SI
C11	-1.0010	-1.0009	0.0000	SI
C12	-1.0010	-1.0009	0.0001	SI
C13	-1.0010	-1.0010	0.0001	SI
C14	-1.0000	-0.9994	0.0049	SI
C15	-1.0010	-1.0010	0.0000	SI
C16	-1.0010	-1.0010	0.0000	SI
C17	-1.0010	-1.0009	0.0000	SI
C18	-1.0010	-1.0010	0.0000	SI
C19	-1.0001	-1.0008	0.0000	SI
C20	-1.0010	-0.9992	0.0046	SI

Tabla G.17: Resultados de LHD para g4.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-30,530.6508	-30,512.3485	718.3319	SI
C2	-30,665.6556	-30,659.5697	10.0681	NO
C3	-30,664.9077	-30,302.6596	1,078.6062	SI
C4	-30,665.6599	-30,665.6484	0.0502	NO
C5	-30,655.6964	-30,612.6386	31.0682	SI
C6	-30,665.0025	-30,621.3112	44.3390	SI
C7	-30,665.6599	-30,665.6420	0.0670	NO
C8	-30,665.6410	-30,639.9002	48.6592	NO
C9	-30,664.8666	-29,313.9184	1,695.5006	SI
C10	-30,636.7945	-30,586.4035	49.7434	SI
C11	-30,665.6599	-30,665.3186	0.8642	NO
C12	-30,665.6599	-30,665.5827	0.1139	NO
C13	-30,665.6071	-30,630.1191	35.4994	NO
C14	-30,570.9007	-29,024.3065	1,475.0482	SI
C15	-30,665.6590	-30,663.6590	2.6183	NO
C16	-30,665.6599	-30,665.6599	0.0000	NO
C17	-30,664.6692	-30,656.9520	10.1658	SI
C18	-30,665.6599	-30,665.6594	0.0027	NO
C19	-30,665.5190	-30,664.3503	2.7926	SI
C20	-30,655.3954	-30,592.0399	47.8572	SI

Tabla G.18: Resultados de LHD para g5.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	5,508.2913	5,244.5690	169.9278	SI
C2	5,126.5300	5,336.8730	255.4984	SI
C3	5,152.6487	5,313.6639	252.1556	SI
C4	5,127.9325	5,344.1199	240.6863	SI
C5	5,172.3845	5,240.3603	205.5770	SI
C6	5,736.4260	5,309.7769	234.4704	SI
C7	5,140.9200	5,287.9866	185.3779	SI
C8	5,678.2949	5,373.9170	273.2471	SI
C9	5,126.9305	5,283.2940	226.9535	SI
C10	5,164.4293	5,450.4604	333.5685	SI
C11	5,131.2578	5,251.2596	163.6110	SI
C12	5,409.4966	5,337.0455	227.2198	SI
C13	5,150.5720	5,295.4289	203.2997	SI
C14	5,136.6495	5,327.6187	235.9133	SI
C15	5,126.5145	5,337.1649	268.4009	SI
C16	5,126.6466	5,354.2830	280.2515	SI
C17	5,129.1354	5,268.1673	146.4232	SI
C18	5,126.8956	5,311.1980	247.7421	SI
C19	5,126.5180	5,342.4367	245.8086	SI
C20	5,134.1956	5,358.9606	284.2900	SI

Tabla G.19: Resultados de LHD para g6.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-6,889.4565	-6,761.8770	122.8734	SI
C2	-6,888.1284	-6,800.1110	98.0023	SI
C3	-6,731.9343	-6,687.8319	72.8873	SI
C4	-6,801.3391	-6,800.1262	25.4556	SI
C5	-6,699.2001	-6,589.3981	102.5639	SI
C6	-6,830.2983	-6,798.8892	145.7899	SI
C7	-6,838.1843	-6,802.1410	108.0233	SI
C8	-6,741.9334	-6,677.3190	102.7373	SI
C9	-6,808.2184	-6,804.1000	108.0025	SI
C10	-6,730.9344	-6,689.3198	62.7311	SI
C11	-6,801.0011	-6,800.1266	24.5162	SI
C12	-6,699.2099	-6,588.3861	105.6530	SI
C13	-6,598.2013	-6,577.9817	42.6329	SI
C14	-6,830.2183	-6,798.8902	175.8290	SI
C15	-6,862.1843	-6,729.4100	188.2311	SI
C16	-6,749.9499	-6,707.1903	62.7030	SI
C17	-6,889.4511	-6,769.0722	128.3114	SI
C18	-6,888.1843	-6,800.0010	94.0243	SI
C19	-6,959.8130	-6,845.1221	110.5439	SI
C20	-6,788.3022	-6,709.2056	91.0992	SI

Tabla G.20: Resultados de LHD para g7.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	24.9986	25.1232	0.8755	SI
C2	25.8778	25.9989	0.1002	SI
C3	24.4994	25.1292	0.7833	SI
C4	25.1198	25.4989	0.3898	SI
C5	26.4993	26.5001	0.1165	SI
C6	25.4993	25.6620	0.2773	SI
C7	24.9935	25.0452	0.1688	SI
C8	24.4749	25.3295	0.8133	SI
C9	25.1598	25.9109	0.8198	SI
C10	26.1003	26.5017	0.4657	SI
C11	25.1178	25.0289	0.1299	SI
C12	24.8014	25.2952	0.5031	SI
C13	25.2194	25.4449	0.2890	SI
C14	27.3093	27.5771	0.2652	SI
C15	25.7191	25.8200	0.1771	SI
C16	24.9991	25.2993	0.3931	SI
C17	25.6632	25.7768	0.2012	SI
C18	25.9923	26.3003	0.4035	SI
C19	24.3180	24.8966	0.6659	SI
C20	25.4428	25.8554	0.5022	SI

Tabla G.21: Resultados de LHD para g8.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-0.095	-0.096	0.0110	SI
C2	-0.096	-0.099	0.0130	SI
C3	-0.095	-0.095	0.0000	SI
C4	-0.095	-0.095	0.0000	SI
C5	-0.096	-0.099	0.0130	SI
C6	-0.095	-0.095	0.0000	SI
C7	-0.095	-0.095	0.0000	SI
C8	-0.095	-0.095	0.0000	SI
C9	-0.095	-0.095	0.0000	SI
C10	-0.095	-0.095	0.0000	SI
C11	-0.095	-0.095	0.0000	SI
C12	-0.095	-0.095	0.0000	SI
C13	-0.095	-0.095	0.0000	SI
C14	-0.095	-0.095	0.0000	SI
C15	-0.095	-0.095	0.0000	SI
C16	-0.095	-0.095	0.0000	SI
C17	-0.095	-0.095	0.0000	SI
C18	-0.095	-0.095	0.0000	SI
C19	-0.095	-0.095	0.0000	SI
C20	-0.095	-0.095	0.0000	SI

Tabla G.22: Resultados de LHD para g9.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	680.6331	684.6324	13.0912	SI
C2	680.6385	680.8076	0.2035	SI
C3	680.6360	680.8916	1.2109	SI
C4	680.6440	680.7136	0.0740	SI
C5	680.6388	683.8966	4.0136	SI
C6	680.6537	681.1135	0.5066	SI
C7	680.6387	680.6542	0.0236	SI
C8	680.6395	680.8742	0.4175	SI
C9	680.6375	680.7157	0.1507	SI
C10	681.0954	682.6099	1.8664	SI
C11	680.6321	680.6769	0.0530	SI
C12	680.6320	680.6592	0.0312	SI
C13	680.6615	681.2659	0.8069	SI
C14	680.6318	888.6992	1,112.8696	SI
C15	680.6307	680.6771	0.0551	SI
C16	680.6326	680.6579	0.0150	SI
C17	680.6353	680.8319	0.5821	SI
C18	680.6317	680.6643	0.0323	SI
C19	680.6322	680.7042	0.0919	SI
C20	680.9028	683.0094	2.5146	SI

Tabla G.23: Resultados de LHD para g10.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	7,544.3851	8,806.0335	2,945.4639	SI
C2	7,048.8614	7,577.0914	677.9138	NO
C3	7,363.0025	8,504.5169	2,373.2151	SI
C4	7,249.2725	7,674.7474	360.4433	SI
C5	7,199.6329	7,769.3963	569.2151	SI
C6	7,097.3329	7,854.8630	486.7218	SI
C7	7,150.7364	7,500.0635	311.1621	SI
C8	7,065.8861	7,660.6152	624.4892	SI
C9	7,513.4878	8,178.4131	2,063.4394	SI
C10	7,319.2610	7,914.2536	458.0747	SI
C11	7,089.2602	7,526.7535	366.6306	SI
C12	7,219.1400	7,519.5633	317.3311	SI
C13	7,099.0142	7,857.2386	548.7240	SI
C14	7,491.0569	8,410.4813	2,698.9168	SI
C15	7,106.9676	7,395.8164	240.4971	SI
C16	7,161.4732	7,401.3607	168.7812	SI
C17	7,106.2766	7,710.6792	499.1047	SI
C18	7,095.5026	7,540.8493	455.4867	SI
C19	7,059.2857	7,499.3537	408.5516	SI
C20	7,165.6973	8,009.1072	750.6504	SI

Tabla G.24: Resultados de LHD para g11.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	0.7500	0.7499	0.0001	SI
C2	0.7498	0.7498	0.0000	SI
C3	0.7499	0.7499	0.0001	SI
C4	0.7498	0.7498	0.0000	SI
C5	0.7498	0.7498	0.0000	SI
C6	0.7498	0.7498	0.0001	SI
C7	0.7498	0.7498	0.0000	SI
C8	0.7498	0.7498	0.0000	SI
C9	0.7500	0.7500	0.0001	SI
C10	0.7498	0.7500	0.0003	SI
C11	0.7498	0.7498	0.0000	SI
C12	0.7498	0.7498	0.0000	SI
C13	0.7498	0.7498	0.0000	SI
C14	0.7500	0.7500	0.0001	SI
C15	0.7498	0.7498	0.0000	SI
C16	0.7498	0.7498	0.0000	SI
C17	0.7498	0.7498	0.0000	SI
C18	0.7498	0.7498	0.0000	SI
C19	0.7498	0.7498	0.0000	SI
C20	0.7498	0.7500	0.0003	SI

Tabla G.25: Resultados de LHD para g12.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-1.0000	-1.0000	0.0000	SI
C2	-1.0000	-1.0000	0.0000	SI
C3	-1.0000	-1.0000	0.0000	SI
C4	-1.0000	-1.0000	0.0000	SI
C5	-1.0000	-1.0000	0.0000	SI
C6	-1.0000	-1.0000	0.0000	SI
C7	-1.0000	-1.0000	0.0000	SI
C8	-1.0000	-1.0000	0.0000	SI
C9	-1.0000	-1.0000	0.0000	SI
C10	-1.0000	-1.0000	0.0000	SI
C11	-1.0000	-1.0000	0.0000	SI
C12	-1.0000	-1.0000	0.0000	SI
C13	-1.0000	-1.0000	0.0000	SI
C14	-1.0000	-1.0000	0.0000	SI
C15	-1.0000	-1.0000	0.0000	SI
C16	-1.0000	-1.0000	0.0000	SI
C17	-1.0000	-1.0000	0.0000	SI
C18	-1.0000	-1.0000	0.0000	SI
C19	-1.0000	-1.0000	0.0000	SI
C20	-1.0000	-1.0000	0.0000	SI

Tabla G.26: Resultados de LHD para g13.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	1.1073	0.5671	0.2972	SI
C2	0.1229	0.6222	0.6259	SI
C3	0.8766	0.5604	0.3721	SI
C4	0.3197	0.8344	0.9079	SI
C5	0.6484	1.5734	3.3498	SI
C6	0.7319	1.5703	2.8833	SI
C7	0.0854	0.6418	0.5280	SI
C8	0.1068	0.9182	1.2535	SI
C9	0.7746	1.2006	3.3853	SI
C10	0.0553	1.0992	1.2398	SI
C11	0.1814	0.6213	0.6296	SI
C12	0.3305	0.8550	0.9284	SI
C13	0.3409	0.7046	0.2958	SI
C14	0.3513	0.5056	0.2437	SI
C15	0.1120	1.0208	2.7209	SI
C16	0.0667	0.4608	0.2089	SI
C17	0.1088	0.7885	1.0441	SI
C18	0.2395	0.7114	0.5033	SI
C19	0.0549	0.8535	1.2014	SI
C20	0.6676	1.4264	2.6425	SI

Tabla G.27: Resultados de LHD para g14.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-31.4337	-13.2756	8.8200	SI
C2	-32.6226	-8.9274	7.8020	SI
C3	-40.8682	-11.6045	9.4893	SI
C4	-27.4340	-7.1559	6.9954	SI
C5	-33.9459	-5.1917	6.2601	SI
C6	-17.7721	-6.8554	5.0870	SI
C7	-32.4073	-8.8979	9.1312	SI
C8	-27.7443	-5.0342	6.5459	SI
C9	-37.4897	-11.3893	9.7616	SI
C10	-29.1815	-18.1485	6.4446	SI
C11	-31.5073	-8.7383	7.9336	SI
C12	-26.3232	-6.7825	6.2338	SI
C13	-34.5891	-7.7994	8.6120	SI
C14	-34.7885	-12.5271	8.9353	SI
C15	-34.5437	-6.2535	7.4490	SI
C16	-33.9061	-12.3119	8.7731	SI
C17	-16.1285	-4.7327	4.7162	SI
C18	-30.3099	-10.6748	9.4276	SI
C19	-42.8663	-22.7584	9.3169	SI
C20	-24.8954	-4.7645	4.6121	SI

Tabla G.28: Resultados de LHD para g15.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	961.7043	961.7700	0.1394	NO
C2	961.6015	961.7444	0.0594	NO
C3	961.7157	961.7656	0.1595	SI
C4	961.7120	961.7445	0.0649	NO
C5	961.7150	961.8895	0.2416	SI
C6	961.7159	961.8441	0.2567	SI
C7	961.5930	961.7179	0.0633	NO
C8	961.7149	961.8164	0.1962	NO
C9	961.7158	961.7346	0.0729	SI
C10	961.7199	962.0440	0.2718	SI
C11	961.7149	961.9647	0.6603	NO
C12	961.6243	961.7107	0.0233	NO
C13	961.7149	961.9031	0.3238	NO
C14	961.7180	964.6269	6.7689	SI
C15	961.6000	961.7214	0.0413	NO
C16	961.6810	961.7395	0.0822	NO
C17	961.6761	961.7337	0.0430	NO
C18	961.5739	961.7051	0.0381	NO
C19	961.7150	961.7372	0.0722	SI
C20	961.7156	962.2430	0.6505	SI

Tabla G.29: Resultados de LHD para g16.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-1.8782	-1.8840	0.0188	SI
C2	-1.9052	-1.8927	0.0177	SI
C3	-1.8374	-1.8072	0.1675	SI
C4	-1.9052	-1.8951	0.0196	SI
C5	-1.9030	-1.8496	0.0590	SI
C6	-1.9051	-1.8585	0.0328	SI
C7	-1.9052	-1.9048	0.0013	SI
C8	-1.9052	-1.8825	0.0290	SI
C9	-1.7229	-1.7251	0.2075	SI
C10	-1.8937	-1.8017	0.0577	SI
C11	-1.9052	-1.8915	0.0269	SI
C12	-1.9052	-1.9051	0.0003	SI
C13	-1.9036	-1.8569	0.0542	SI
C14	-1.3629	-1.7346	0.1934	SI
C15	-1.9052	-1.8817	0.0247	SI
C16	-1.9052	-1.9052	0.0000	SI
C17	-1.9052	-1.8956	0.0108	SI
C18	-1.9052	-1.8980	0.0171	SI
C19	-1.9052	-1.9052	0.0000	SI
C20	-1.8779	-1.7022	0.1598	SI

Tabla G.30: Resultados de LHD para g17.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	8,927.5977	8,891.8350	78.9226	SI
C2	8,853.5395	8,862.3847	21.8980	SI
C3	8,853.5436	8,879.3093	40.3352	SI
C4	8,853.5395	8,865.1476	24.6254	SI
C5	8,853.8982	8,859.8055	13.0455	SI
C6	8,853.5395	8,857.0699	9.5127	SI
C7	8,853.5404	8,860.6448	18.7215	SI
C8	8,853.5395	8,865.1881	24.6688	SI
C9	8,855.9398	8,933.3512	85.5636	SI
C10	8,854.2928	8,859.7367	3.7444	SI
C11	8,853.5398	8,910.8274	30.4156	SI
C12	8,853.5401	8,856.5005	13.2256	SI
C13	8,853.5460	8,864.4707	21.8043	SI
C14	8,853.5433	8,967.4879	115.8160	SI
C15	8,853.5395	8,864.7357	24.7319	SI
C16	8,853.5395	8,878.6063	34.6689	SI
C17	8,853.5474	8,860.2804	18.1242	SI
C18	8,853.5395	8,864.5729	24.7680	SI
C19	8,853.5395	8,866.4014	27.3828	SI
C20	8,853.8088	8,865.8104	22.2341	SI

Tabla G.31: Resultados de LHD para g18.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	-0.8559	-0.7660	0.1318	SI
C2	-0.8661	-0.7918	0.1161	SI
C3	-0.6601	-0.7498	0.1040	SI
C4	-0.8661	-0.8084	0.0867	SI
C5	-0.8661	-0.7520	0.1079	SI
C6	-0.8657	-0.7602	0.0967	SI
C7	-0.8661	-0.7714	0.1039	SI
C8	-0.8659	-0.7533	0.1181	SI
C9	-0.8604	-0.7380	0.1176	SI
C10	-0.8648	-0.6622	0.1672	SI
C11	-0.8661	-0.8034	0.0937	SI
C12	-0.8659	-0.7885	0.0932	SI
C13	-0.8659	-0.8034	0.1081	SI
C14	-0.5349	-0.7720	0.1047	SI
C15	-0.8661	-0.7917	0.1013	SI
C16	-0.8660	-0.7894	0.0890	SI
C17	-0.8660	-0.8078	0.0945	SI
C18	-0.8661	-0.8244	0.0807	SI
C19	-0.8658	-0.8121	0.0746	SI
C20	-0.8660	-0.9221	0.0986	SI

Tabla G.32: Resultados de LHD para g19.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	48.8471	177.5650	147.1668	SI
C2	37.8446	67.8172	15.0011	SI
C3	42.1369	99.5141	102.3667	SI
C4	41.1965	62.8130	12.7159	SI
C5	49.1941	87.8779	25.0471	SI
C6	43.2908	81.0956	25.2906	SI
C7	38.0125	55.8556	11.9744	SI
C8	42.6472	64.9762	13.5596	SI
C9	41.6670	78.7204	73.4272	SI
C10	69.2878	91.4919	15.9576	SI
C11	39.6310	53.8740	7.5095	SI
C12	42.4817	55.9715	8.6344	SI
C13	42.2087	76.0695	16.2010	SI
C14	45.3211	107.1562	103.4789	SI
C15	42.2853	55.1476	8.9822	SI
C16	39.8917	52.9636	10.2889	SI
C17	38.9800	55.3726	10.8973	SI
C18	42.0602	60.2588	13.6925	SI
C19	41.9183	53.8208	11.2927	SI
C20	40.0137	97.3141	40.2594	SI

G.3. Funciones de ingeniería estructural

Tabla G.33: Resultados de LHD para t1.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	4,657.3173	4,666.2389	6.9006	SI
C2	4,658.0341	4,665.9487	6.6346	SI
C3	4,657.1478	4,661.9220	2.8560	SI
C4	4,658.9437	4,664.5800	3.4520	SI
C5	4,661.7901	4,703.4236	79.7820	SI
C6	4,661.3776	4,675.4808	20.3689	SI
C7	4,657.8624	4,662.9288	3.3006	SI
C8	4,665.4539	4,919.0487	237.4501	SI
C9	4,657.4206	4,661.2550	2.9050	SI
C10	5,814.5745	6,565.7499	270.4850	SI
C11	4,656.8460	4,661.6940	3.1255	SI
C12	4,658.0155	4,666.5438	9.5674	SI
C13	4,659.7330	4,917.3808	253.4836	SI
C14	4,657.3966	4,663.9745	5.9339	SI
C15	4,658.8222	4,674.1102	7.0500	SI
C16	4,656.9274	4,661.7870	4.1421	SI
C17	4,657.7205	4,667.6191	8.2530	SI
C18	4,657.9522	4,662.4380	6.4900	SI
C19	4,657.8720	4,661.5809	2.3820	SI
C20	6,934.7654	9,123.6767	1,265.3043	SI

Tabla G.34: Resultados de LHD para t2.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	467.3281	473.2014	11.7050	SI
C2	467.3161	467.7727	0.7036	SI
C3	467.3161	468.0840	0.9527	SI
C4	467.7997	481.5809	14.7905	SI
C5	467.4852	472.1294	5.2809	SI
C6	467.3259	470.1045	3.8744	SI
C7	471.6802	506.0536	22.9112	SI
C8	467.3115	467.8321	0.7229	SI
C9	500.4773	532.6138	15.8139	SI
C10	467.3075	469.4459	9.1641	SI
C11	467.3066	469.0071	5.9711	SI
C12	468.2969	488.5022	21.6024	SI
C13	467.3127	468.8528	1.0319	SI
C14	467.3138	468.7729	2.6704	SI
C15	467.3292	468.3319	1.0222	SI
C16	467.3211	468.5070	2.4001	SI
C17	467.3077	467.6805	1.4013	SI
C18	467.3611	468.2863	1.2455	SI
C19	548.6575	723.6022	117.9235	SI
C20	470.6666	469.9100	0.3280	SI

Tabla G.35: Resultados de LHD para t3.

Configuración	Mejor	Media	Desv. Estándar	¿Factible?
C1	12,122.6819	14,101.1458	1,807.3159	NO
C2	11,650.5999	13,322.6593	1,280.1255	NO
C3	11,470.1916	12,621.4344	857.8105	NO
C4	11,525.3524	13,015.4139	937.6090	NO
C5	12,138.1965	13,807.1116	1,178.4309	NO
C6	11,808.4884	13,418.9719	1,042.6224	NO
C7	11,272.3857	12,659.5594	1,109.9731	NO
C8	11,619.0749	15,998.1953	2,780.8144	NO
C9	11,455.7420	12,522.1693	605.0521	NO
C10	23,433.6909	26,278.8871	1,779.2604	SI
C11	22,737.6156	26,637.2428	1,938.3251	SI
C12	11,126.0698	12,201.8628	796.6547	NO
C13	12,188.3431	14,970.5897	1,648.6951	NO
C14	11,785.7326	12,897.3891	866.8708	NO
C15	12,236.3671	14,903.3914	1,307.1288	NO
C16	11,149.1575	12,673.7666	876.1248	NO
C17	23,776.3178	26,505.8846	1,565.3013	SI
C18	11,053.4871	13,085.2369	988.9204	NO
C19	11,324.7888	12,288.2398	535.3832	NO
C20	11,201.0134	13,280.5107	1,250.7023	NO

Bibliografía

[1] Abido, M. A.: *Optimal Power Flow using Particle Swarm Optimization*. International Journal of Electrical Power and Energy Systems, 24(7):563–571, 2002.

[2] Adeli, H. y Cheng, N.: *Augmented Lagrangian Genetic Algorithm for Structural Optimization*. Journal of Aerospace Engineering, 7(1):104–118, 1994.

[3] Álvarez Benítez, J. E., Everson, R. M. y Fieldsend, J. E.: *A MOPSO Algorithm Based Exclusively on Pareto Dominance Concepts*. En *Evolutionary Multicriterion Optimization, EMO 2005*, páginas 459–473, 2005. Lecture Notes in Computer Science, vol. 3410.

[4] Angeline, P. J.: *Evolutionary Optimization versus Particle Swarm Optimization: Philosophy and Performance Differences*. Evolutionary Programming VII, 7th International Conference, EP 98 - Lectures Notes in Computer Science, 1447, 1998. V. W. Porto, N. Saravanan, D. Waagen and A. E. Eiben (Eds), Springer.

[5] Arora, J.: *Introduction to Optimum Design*. McGraw-Hill, 1989.

[6] Bäck, T., Fogel, D. B. y Michalewicz, Z.: *Evolutionary Computation 1: Basic Algorithms and Operators*. IOP Press, 2000.

[7] Bäck, T., Fogel, D. B. y Michalewicz, Z.: *Evolutionary Computation 2: Advanced Algorithms and Operators*. IOP Press, 2000.

[8] Bakirtzis, A., Petridis, V. y Kazarlis, S.: *Genetic algorithm solution to the economic Dispatch problem*. Proc. Inst. Elect. Eng., Gener. Transm., Distrib., 141(4):377–382, July 1994.

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

[10] Belegundu, A.: *A Study of Mathematical Programming Methods for Structural Optimization*. Tesis de Doctorado, Department of Civil Environmental Engineering, University of Iowa, 1982.

[11] Bernardino, H.S., Barbosa, H.J.C. y Lemong, A.C.C: *A Hybrid Genetic Algorithm for Constrained Optimization Problems in Mechanical Engineering*. En *2007 IEEE Congress on Evolutionary Computation (CEC'2007)*, páginas 646–653, Singapore, 2007. IEEE Press.

[12] Blackwell, T. y Branke, J.: *Multiswarms, exclusion, and anti-convergence in dynamic environments.* IEEE Transactions on Evolutionary Computation, 10(4):459–472, 2006.

[13] Branke, J. y Mostaghim, S.: *About Selecting the Personal Best in Multi-Objective Particle Swarm Optimization.* En Runarsson, Thomas Philip, Beyer, Hans Georg, Burke, Edmund K., Guervós, Juan J. Merelo, Whitley, L. Darrell y Yao, Xin (editores): *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, volumen 4193 de *Lecture Notes in Computer Science*, páginas 523–532. Springer, 2006, ISBN 3-540-38990-3.

[14] Cagnina, L., Esquivel, S. y Gallard, R.: *Particle Swarm Optimization for sequencing problems: a case study.* En *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC'2004)*, páginas 536–541, Portland, Oregon, USA, 2004.

[15] Cagnina, L. C., Errecalde, M. L., Ingaramo, D. A. y Rosso, P.: *A Discrete Particle Swarm Optimizer for Clustering Short-Text Corpora.* En Bogdan Filipič y Jurij Šilc (editores): *Proceedings of the Third International Conference on Bioinspired Optimization Methods and their Applications, BIOMA 2008*, páginas 93–103. Jožef Stefan Institute, Ljubljana, Eslovenia, 2008.

[16] Cagnina, L. C., Esquivel, S. C. y Coello Coello, C.: *Solving Engineering Optimization Problems with the Simple Constrained PSO.* En Informatika, Slovene Society (editor): *Informatica - International Journal of Computing and Informatics*, volumen 32, páginas 319–326, 2008. ISSN: 0350-5596.

[17] Cagnina, L. C., Esquivel, S. C. y Coello Coello, C. A.: *A Particle Swarm Optimizer for Constrained Numerical Optimization.* En Runarsson, Thomas Philip, Beyer, Hans Georg, Burke, Edmund, Merelo-Guervós, Juan J., Whitley, L. Darrell y Yao, Xin (editores): *Parallel Problem Solving from Nature (PPSN IX). 9th International Conference*, páginas 910–919, Reykjavik, Iceland, 2006. Reykjavik, Iceland, Springer. Lecture Notes in Computer Science Vol. 4193.

[18] Cao, Y. y Wu, Q.: *Mechanical Design Optimization by Mixed-variable Evolutionary Programming.* En *1997 IEEE International Conference on Evolutionary Computation*, páginas 443–446, 1997.

[19] Carlisle, A. y Dozier, G.: *An off-the-shelf PSO.* En *Proceedings of the Workshop on Particle Swarm Optimization*, páginas 1–6, 2001.

[20] Cha, J. y Mayne, R.: *Optimization with Discrete Variables via Recursive Quadratic Programming: part II.* Transaction of ASME, 111:130–136, 1989.

[21] Chakravarti, I. M., Laha, R. G. y Roy, J.: *Handbook of Methods of Applied Statistics*, volumen 1. John Wiley, 1967.

[22] Chambers, J., Cleveland, W., Kleiner, B. y Tukey, P.: *Graphical Methods for Data Analysis.* Wadsworth, Belmont CA, 1983.

[23] Charnes, A. y Cooper, W. W.: *Management Models and Industrial Applications of Linear Programming*, volumen 1. John Wiley, 1961.

[24] Clerc, M.: *The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization*. En *Proceeding of IEEE Congress on Evolutionary Computation*, volumen 3, páginas 1957–1999, 1999.

[25] Coath, G. y Halgamuge, S. K.: *A Comparison of Constraint-Handling Methods for the Application of Particle Swarm Optimization to Constrained Non-linear Optimization Problems*. En *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'2003)*, páginas 2419–2425, 2003.

[26] Coello Coello, C., Lamont, G. y Van Veldhuizen, D.: *Evolutionary Algorithms for Solving Multi-objective Problems*. Springer, 2007. ISBN 978-0-387-33254-3.

[27] Coello Coello, C. A.: *Theorical and Numerical Constraint Handling Techniques used with Evolutionary Algorithms: A Survey of the State if the Art*. Computer Methods in Applied Mechanics and Engineering, 191(11), 2002.

[28] Coello Coello, C. A., Luna, E. H. y Aguirre, A. H.: *Use of Particle Swarm Optimization to Design Combinational Logic Circuits*. Lecture Notes in Computer Science, 2606:398–409, 2003. Springer-Verlag.

[29] Coello Coello, C. A., Rudnick, M. y Christiansen, A. D.: *Using Genetic Algorithms for Optimal Design of Trusses*. En *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, páginas 88–94, New Orleans, Louisiana, USA, 1994. IEEE Computer Society Press.

[30] Cohon, J. L.: *Multiobjective Programming and Planning*. Academic Press, 1978.

[31] Cohon, J. L. y Marks, D. H.: *A Review and Evaluation of Multiobjective Programming Techniques*. Water Resources Research, 11(2):208–220, 1975.

[32] Daneshyari, M. y Yen, G. G.: *Diversity-based Information Exchange among Multiple Swarms in Particle Swarm Optimization*. En *2006 IEEE Congress on Evolutionary Computation (CEC'2006)*, páginas 1686–1693, Vancouver, BC, Canada, 2006. IEEE Press.

[33] Dasgupta, D. (editor): *Artificial Inmune Systems and Their Applications*. Springer-Verlag, Berlin, 1999.

[34] Davis, L.: *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[35] Deb, K., Agrawal, S., Pratap, A. y Meyarivan, T.: *A fast and elitist multiobjective genetic algorithm: NSGA-II*. IEEE Trans. Evolutionary Computation, 6(2):182–197, 2002.

[36] Deb, K., Thiele, L., Laumanns, M. y Zitzler, Z.: *Scalable Multi-Objective Optimization Test Problems.* En *Congress on Evolutionary Computation (CEC 2002)*, volumen 1, páginas 825–830, Piscataway, New Jersey, 2002. IEEE Service Center.

[37] Durillo, J. J., García Nieto, J., Nebro, A. J., Coello Coello, C. A., Luna, F. y Alba, E.: *Multi-Objective Particle Swarm Optimizers: An Experimental Comparison.* En *Evolutionary Multicriterion Optimization, 5th International Conference EMO 2009*, páginas 495–509, 2009. Lecture Notes in Computer Science, vol. 5467.

[38] Durillo, J. J., Nebro, A. J., Luna, F., Dorronsoro, B. y Alba, E.: *jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics.* Informe técnico ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, 2006. http://jmetal.sourceforge.net/.

[39] Eberhart, R. y Kennedy, J.: *A New Optimizer using Particle Swarm Theory.* En *Proceeding of the Sixth International Symposium on Micromachine and Human Science*, página 39, 1995. Nagoya, Japan.

[40] Eberhart, R. y Kennedy, J.: *A discrete version of the Particle Swarm Algorithm.* En *Proceeding of the World Multiconference on Systemics, Cybernetics and Informatics 1007*, páginas 4104–4109, 1997. Picataway, NJ.

[41] Eberhart, R. C. y Hu, X.: *Human Tremor Analysis using Particle Swarm Optimization.* En *Proceeding of the IEEE Congress on Evolutionary Computation*, volumen 3, páginas 1927–1930, 1999.

[42] Eberhart, R. C., Simpson, P. K. y Dobbins, R. W.: *Computational Intelligence PC Tools.* En *Academic Press Professional Press*, 1996. 1era edición.

[43] Eggermont, J., Eiben, A. y Hemert, J.: *Adapting the fitness function in GP for Data Mining.* En Springer-Verlag (editor): *Proceedings of the Euro Genetic Programming Conference*, volumen 1598, páginas 193–202, 1999.

[44] El-Gallad, A. I., El-Hawary, M. E., Sallam, A. A. y Kalas, A.: *A Particle Swarm Optimizer for Constrained Economic Dispatch with Prohibited Operating Zones.* En *Canadian Conference on Electrical and Computer Engineering*, páginas 78–81, 2002.

[45] El-Sayed, M. y Jang, T.: *Structural optimization using unconstrained non-linear goal programming algorithm.* Computers and Structures, 52(4):723–727, 1994.

[46] Engelbrecht, A.: *Fundamentals of Computational Swarm Intelligence.* John Wiley & Sons Ltd., 2005.

[47] Esquivel, S. y Gallard, R.: *Heurísticas de Búsqueda Local y Técnicas de Hibridización.* Material Didáctico 7, 1999.

[48] Esquivel, S. C. y Cagnina, L. C.: *Multi-Objective Optimization with a Gaussian PSO algorithm.* En *XIV Congreso Argentino en Ciencias de la Computación (CACIC 2008)*, páginas 401–412, 2008. ISBN: 978-987-24611-0-2.

[49] Fan, J. Y. y Zhang, L.: *Real-time economic Dispatch with line flow and emission constraints using quadratic programming.* IEEE Trans. Power Syst., 13(2):320–325, May 1998.

[50] Fisher, R. A.: *The Design of Experiments.* Oliver and Boyd, Edinburgh, 1935.

[51] Fogel, L., Owens, A. y Walsh, M.: *Artificial Intelligence through Simulated Evolution.* Wiley, New York, 1966.

[52] Franken, N. y Engelbrecht, A. P.: *PSO approaches to Co-Evolve IPD Strategies.* En *IEEE Congress on Evolutionary Computation*, páginas 356–363, 2004.

[53] Freitas, A.: *A Genetic programming framework for two data mining tasks: classification and generalized rule induction.* En Koza, J., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Iba, H. y Riolo, R. (editores): *Proceedings of the Second Annual Conference on Genetic Programming*, páginas 96–101. Morgan Kaufmann, 1997.

[54] Galante, M.: *Structures Optimization by a simple genetic algorithm.* En *Numerical methods in engineering and applied sciences*, páginas 862–870, Barcelona, Spain, 1992.

[55] Gardner, M.: *Mathematical games: on the remarkable császár polyhedron and its applications in problem solving.* En *Sci. Amer.*, páginas 102–107, 1975.

[56] Gere, J. M. y Weaver, W.: *Analysis of Framed Structures.* D. Van Nostrand Company, Inc., 1965.

[57] Gies, D. y Rahmat-Samii, Y.: *Reconfigurable Antenna Array Design using Paralell PSO.* En *Proceedings of the IEEE Society International Conference on Antennas and Propagation*, páginas 177–180, 2003.

[58] Glover, F. y Laguna, M.: *Tabú Search*, 1997. Kluwer, London.

[59] Goicoechea, A.: *A Multi-objective Stochastic Programming Model in Watershed Management.* Informe técnico, Dept. of system and industrial engineering, University of Arizona, Tucson, Arizona, 1977. No publicado.

[60] Goicoechea, A., Duckstein, L. y Fogel, M.: *Multiple Objectives under Uncertainty: an Illustrative Application of PROTRADE.* Water Resources Research, 15(2):203–210, 1979.

[61] Goldberg, D. E. y Deb, K.: *A comparison of selection schemes used in genetic algorithms.* En Rawlins, G. J. E. (editor): *Foundations of Genetic Algorithms*, páginas 69–93. Morgan Kaufmann, 1991.

[62] Golinski, J.: *An Adaptive Optimization System Applied to Machine Synthesis.* Mechanism and Machine Synthesis, 8:1973, 1973.

[63] Guerequeta, R. y Vallecillo, A.: *Técnicas de Diseño de Algoritmos.* Servicio de Publicaciones de la Universidad de Málaga, 1998. ISBN=84-7496-666-3, Dirección temática: http://www.lcc.uma.es/∼av/Libro/CAP5.pdf.

[64] Haimes, Y. Y., Lasdon, L. S. y Wismer, D. A.: *On a bicriterion Formulation of the Problems of Integrated System Identification and System Optimization.* IEEE Transaction on Systems, Man, and Cybernetics, 1(3):296–297, 1971.

[65] Hernández-Aguirre, A., Botello-Rionda, S., Coello Coello, C. A., Lizárraga-Lizárraga, G. y Mezura-Montes, E.: *Handling Constraints using Multiobjective Optimization Concepts.* International Journal for Numerical Methods in Engineering, 59(15):1989–2017, 2004.

[66] Hernandez Aguirre, A., Muñoz Zavala, A., Villa Diharce, E. y Botello Rionda, S.: *COPSO: Constrained Optimization via PSO Algorithm.* Informe técnico I-07-04/22-02-2007, Center for Research in Mathematics (CIMAT), 2007.

[67] Hertz, A., Taillard, E. y de Werra, R.: *A Tutorial on Tabu Search.* En *Proceedings of Giornate di Lavoro AIRO 95 (Enterprise Systems: Management of Technical and Organizational Changes)*, páginas 13–24, 1995.

[68] Hindi, K. S. y Ghani, M. R. A.: *Dynamic economic dispatch for large scale power systems: a Lagrangian relaxation approach.* Journal of Electrical Power & Energy Systems, 13:51–56, Feb 1991.

[69] Hirata, N., Ishigame, A. y Nishigaito, N.: *Neuro Stabilizing Control Based on Lyapunov Method for Power System.* En *SICE 2002. Proceeding of the 41st SICE Annual Conference*, volumen 5, páginas 3169–3171, 2002.

[70] Holland, J. H.: *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[71] Hollander, M. y Wolfe, D. A.: *Nonparametric Statistical Methods.* Wiley, 1973.

[72] Horn, J.: *Multicriterion Decision Making.* Handbook of Evolutionary Computation, 1:F1.9:1–F1.9:15, 1997.

[73] http://es.wikipedia.org/wiki/Algoritmo_divide_y_venceras.

[74] Hu, X., Eberhart, R. y Shi, Y.: *Swarm Intelligence for Permutation Optimization: a Case Study on n-queens Problem.* En *Proceeding of the IEEE Swarm Intelligence Symposium*, páginas 243–246, Indianapolis, Indiana, USA, 2003.

[75] Huang, V. L., Suganthan, P. N. y Liang, J. J.: *Comprehensive Learning Particle Swarm Optimizer for Solving Multiobjective Optimization Problems.* Int. J. Intell. Syst., 21(2):209–226, 2006.

[76] Huband, S., Hingston, P., Barone, L. y While, L.: *A Review of Multiobjective Test Problems and a Scalable Test Problem Toolkit*. IEEE Transactions on Evolutionary Computation, 10(5):477–506, 2006.

[77] Ingaramo, D., Errecalde, M., Cagnina, L. y Rosso, P.: *Particle Swarm Optimization for clustering short-text corpora*, capítulo en: Computational Intelligence and Bioengineering (Essays in Memory of Antonina Starita), páginas 3–19. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009.

[78] Iriji, Y.: *Management Goals and Accounting for Control*. North-Holland, 1965.

[79] J. Fu, R. Fenton y Cleghorn, W.: *A Mixed Integer-discrete-continuous Programming Method and its Applications to Engineering Design Optimization*. Engineering Optimization, 17:263–280, 1991.

[80] Janson, S. y Middendorf, M.: *A Hierarchical Particle Swarm Optimizer and its Adaptive Variant*. En *IEEE Transactions on Systems, Man and Cybernetics - Part B*, volumen 35(6), páginas 1272–1282, 2005.

[81] Keeney, R. J.: *Multi-dimensional Utility Functions: Theory, Assessment and Applications*. Informe técnico 43, Massachusetts Institute of Technology, Operations Research Center, Cambridge, Massachusetts, USA, 1969.

[82] Kennedy, J.: *The Particle Swarm: Social Adaptation of Knowledge*. En *Proceedings of the IEEE International Conference on Evolution Computation*, páginas 303–308, 1997.

[83] Kennedy, J.: *Small World and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance*. En *Proceedings of the 1999 Congress on Evolutionary Computation*, páginas 1931–1938, 1999. Picataway, NJ: IEEE Service Center.

[84] Kennedy, J. y Eberhart, R.: *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[85] Kennedy, J. y Mendes, R.: *Neighborhood Topologies in Fully-Informed and Best-of-Neighborhood Particle Swarm*. En *Proceedings of the IEEE International Workshop on Soft Computing in Industrial Applications*, páginas 45–50, 2003.

[86] Kirkpatrick, S.: *Optimization by Simulated Annealing - Quantitative Studies*. En *Journal of Statistical Physics*, volumen 34, páginas 975–986, 1984.

[87] Knowles, J. D. y Corne, D. W.: *Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy*. Evolutionary Computation, 8(2):149–172, 2000.

[88] Koski, K.: *Multicriterion Optimization in Structural Design*. En E. Atrek, R. H. Gallagher, K. M. Ragsdell y O. C. Zienkiewicz (editor): *New Directions in Optimum Structural Design*, páginas 483–503. John Wiley and Sons, 1984.

[89] Koza, J. R.: *Hierarchical Genetic Algorithms Operating on Populations of Computer Programs*. En *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volumen 1, páginas 768–774, 1989.

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

[91] Koziel, S. y Michalewicz, Z.: *Evolutionary Algorithms, Homomorphous Mappings, and Constrained Optimization*. Evolutionary Computation, 7(1):19–44, 1999.

[92] Krohling, R. A. y Dos Santos Coelho, L.: *PSO-E: Particle Swarm with Exponencial Distribution*. En *2006 IEEE Congress on Evolutionary Computation (CEC'2006)*, páginas 5577–5582, Vancouver, BC, Canada, July 2006. IEEE.

[93] Landa Becerra, R.: *Algoritmos Culturales Aplicados a Optimización con Restricciones y Optimización Multiobjetivo*. Tesis para obtener el grado de Maestro en Ciencias, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Departamento de Computación, México, 2002.

[94] Landa Becerra, R.: *Use of Domain Information to Improve the Performance of an Evolutionary Algorithm*. Tesis de Doctorado, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Departamento de Computación, México, 2007.

[95] Landa Becerra, R. y Coello Coello, C. A.: *Cultured differential evolution for constrained optimization*. Computer Methods in Applied Mechanics and Engineering, 195(33–36):4303–4322, July 1 2006.

[96] Landa Becerra, R. y Coello Coello, C. A.: *Solving Hard Multiobjective Optimization Problems Using ϵ-Constraint with Cultured Differential Evolution*. En *PPSN*, páginas 543–552, 2006.

[97] Laumanns, M., Thiele, L. y Zitzler, E.: *An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method*. European Journal of Operational Research, (169):932–942, 2006.

[98] Lee, K. S. y Geem, Z. W.: *A new structural optimization method based on the harmony search algorithm*. Computers and Structures, 82(9–10):781–798, 2004.

[99] Lemonge, A.C.C. y Barbosa, H.J.C.: *An Adaptive Penalty Scheme for Genetic Algorithms in Structural Optimization*. International Journal for Numerical Methods in Engineering, 59(5):703–736, February 2004.

[100] Li, F., Morgan, R. y Williams, D.: *Hybrid genetic approaches to ramping rate constrained Dynamic economic dispatch*. Electric Power Systems Research, 43:97–103, 1997.

[101] Li, H. y Chou, T.: *A Global Approach of Nonlinear Mixed Discrete Programming in Design Optimization*. Engineering Optimization, 22:109–122, 1994.

[102] Li, L. J., Huang, Z. B., Liu, F. y Wu, Q. H.: *A heuristic particle swarm optimizer for optimization of pin connected structures.* Computers and Structures, 85(7–8):340–349, 2007.

[103] Li, X.: *A Non-dominated Sorting Particle Swarm Optimizer for Multiobjective Optimization.* En *Genetic and Evolutionary Computation, GECCO 2003*, 2003. Vol. 2723 of LNCS, páginas 37-48.

[104] Liang, J. J., Runarsson, T. P., Mezura-Montes, E., Clerc, M., Suganthan, P. N., Coello Coello, C. A. y Deb, K.: *Problem Definitions and Evaluation Criteria for the CEC 2006.* Informe técnico, Nanyang Technological University, Singapore, 2006.

[105] Liang, J. J. y Suganthan, P. N.: *Dynamic multi-swarm particle swarm optimizer.* En *Proceedings of the IEEE 2005 Swarm Intelligence Symposium (SIS 2005)*, páginas 124–129. IEEE Press, 2005.

[106] Lin, C. E. y Viviani, G. L.: *Hierarchical economic Dispatch for piecewise quadratic cost functions.* IEEE Trans. Power App. Syst., PAS-103(6):1170–1175, 1984.

[107] Lin, W. M., Cheng, F. S. y Tsay, M. T.: *An improved Tabu search for economic Dispatch with multiple minima.* IEEE Trans. Power Syst., 17:108–112, 2002.

[108] Liu, D. y Cai, Y.: *Taguchi method for solving the Economic dispatch problem with nonsmooth cost functions.* IEEE Trans. Power Syst., 20:2006–2014, 2005.

[109] Loh, H. y Papalambros, P.: *A Sequential Linearization Approach for Solving Mixed-discrete Nonlinear Design Optimization Problems.* ASME Journal of Mechanical Design, 113:325–334, 1991.

[110] López Ramírez, B. y Mezura-Montes, E.: *Comparación de Algoritmos Evolutivos y Bio-Inspirados en Problemas de Optimización con Restricciones.* En *Memorias del IV Encuentro Participación de la Mujer en la Ciencia*, 2007. ISBN 978-968-9241-03-4.

[111] Lounis, Z. y Cohn, M. Z.: *Multiobjective Optimization of Prestressed Concrete Structures.* Journal of Structural Engineering, 119(3):794–808, 1993.

[112] Lowry, R.: *One Way ANOVA - Independent Samples*, 2008.

[113] Lu, W. Z., Fan, H. Y. y Lo, S. M.: *Application of Evolutionary Neural Network Method in Predicting Pollutant Levels in Downtown Area of Hong Kong.* En *Neurocomputing*, volumen 51, páginas 387–400, 2003.

[114] Mavrotas, G.: *Generation of efficient solutions in Multiobjective Mathematical Programming problems using GAMS. Effective implementation of the ϵ-constraint method.* Informe técnico, Laboratory of Industrial and Energy Economics, School of Chemical Engineering. Mational Technical University of Athenas, 2008.

[115] McKay, M. D., Beckman, R. J. y Conover, W. J.: *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code.* Technometrics, 21(2):239–245, 1979.

[116] McKinley, S. y Levine, M.: *Cubic Spline Interpolation.* Math 45: Linear Algebra, 1998.

[117] Memari, A.M. y Fuladgar, A.: *Minimum Weight Design of Trusses by BEHSAZ Program.* En *Proceedings of the 2nd International Conference on Computational Structures Technology*, Athens, Greece, August 30–September 1 1994.

[118] Mendes, R., Kennedy, J. y Neves, J.: *The Fully Informed Particle Swarm: Simpler maybe Better.* En *IEEE Transactions on Evolution Computation*, volumen 8, páginas 204–210, 2004.

[119] Messerschmidt, L. y Engelbrecht, A. P.: *Learning to Play Games using a PSO-Based Competitive Learning Approach.* En *IEEE Transactions on Evolutionary Computation*, volumen 8(3), páginas 280–288, 2004.

[120] Metropolis, N. y Ulam, S.: *The Monte Carlo Method.* Journal of the American Statistical Association, 44(247):335–341, 1949.

[121] Mezura-Montes, E. y Coello Coello, C. A.: *Useful Infeasible Solutions in Engineering Optimization with Evolutionary Algorithms.* En Gelbukh, A. F., de Albornoz, A. y Terashima-Marín, H. (editores): *MICAI*, páginas 652–662, 2005.

[122] Michalewicz, Z.: *A Survey of Constrained Handling Techniques in Evolutionary Computation Methods.* En *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, páginas 135–155. MIT Press, 1995.

[123] Michalewicz, Z. y Fogel, D. B.: *How to solve it: Modern Heuristics.* Springer-Verlag Berling Heidelberg, New York, 2000. ISBN: 3-540-66061.

[124] Michalewicz, Z. y Janikow, C. Z.: *Handling Constraints in Genetic Algorithms.* En Belew, R. K. y Booker, L. B. (editores): *Proceedings of the Fourth International Conference on Genetic Algorithms*, páginas 151–157, California, 1991. Morgan Kaufmann.

[125] Mohan, C. y Al-Kazemi, B.: *Discrete Particle Swarm Optimization.* En *Proceeding of the Workshop on Particle Swarm Optimization*, Indianapolis, IN, 2001.

[126] Monarchi, D. E., Kisiel, C. C. y Duckstein, L.: *Interactive Multiobjective Programming in Water Resources: a Case Study.* Water Resources Research, 9(4):837–850, 1973.

[127] Moore, J. y Chapman, R.: *Application of Particle Swarm to Multiobjective Optimization*, 1999. No publicado.

[128] Mostaghim, S. y Teich, J.: *Strategies for Finding Good Local Guides in Multi-objective Particle Swarm Optimization.* En *Proceedings of the IEEE Swarm Intelligence Symposium*, páginas 26–33, 2003.

[129] Naka, S., Genji, T., Yura, T. y Fukuyama, Y.: *Practical distribution state estimation using hybrid particle swarm optimization.* Proc. IEEE Power Engineering Society Winter Meeting, 2:815–820, Feb. 2001.

[130] Nebro, A. J., Durillo, J. J., García Nieto, J., Coello Coello, C. A., Luna, F. y Alba, E.: *SMPSO: A New PSO-based Metaheuristic for Multi-objective Optimization.* En *2009 IEEE Sysmposium on Computational Intelligence in Multicriteria Decision-Making*, páginas 66–73, USA, 2009. IEEE Press.

[131] Nunes De Castro, L. y Timmis, J.: *An artificial inmune network for multimodal function optimization.* En *Proceedings of the special sessions on artificial inmune systems in the 2002 Congress on Evolutionary Computation, 2002 IEEE World Congress on Computational Intelligence*, volumen I, páginas 669–674, Honolulu, Hawaii, Mayo 2002.

[132] Okabe, T.: *Evolutionary Multi-Objective Optimization - On the Distribution of Offspring in Parameter and Fitness Space.* Tesis de Doctorado, Bielefeld University, 2004.

[133] Omran, M. G., Engelbrecht, A. P. y Salman, A.: *Particle Swarm Optimization Method for Image Clustering.* En *International Journal on Pattern Recognition and Artificial Intelligence*, páginas 297–322, 2005. 19(3).

[134] Ongsakul, W. y Ruangpayoongsak, N.: *Constrained dynamic economic dispatch by simulated annealing/genetic algorithms.* En *IEEE Power Engineering International Conference on Power Industry Computer Applications (PICA 2001)*, páginas 207–212. IEEE Press, 2001.

[135] Onwubolu, G. C. y Clerc, M.: *Optimal Path for Automated Drilling Operations by a New Heuristic Approach using Particle Swarm Optimization.* International Journal of Production Research, 4:473–491, 2004.

[136] Osyczka, A.: *Multicriterion Optimization in Engineering with FORTRAN Programs.* Ellis Horwood Limited, 1984.

[137] Osyczka, A.: *Multicriteria Optimization for Engineering Design.* En Gero, J. S. (editor): *Design Optimization*, páginas 193–227. Academic Press, 1985.

[138] Park, J. B., Lee, K. S., Shin, J. R. y Lee, K. Y.: *A particle swarm optimization for Economic dispatch with nonsmooth cost functions.* IEEE Trans. Power Syst., 20:34–42, 2005.

[139] Park, J. H., Kim, Y. S., Eom, I. K. y Lee, K. Y.: *Economic load dispatch for piecewise quadratic cost function using Hopfield neural network.* IEEE Trans. Power Syst., 8:1030–1038, 1993.

[140] Park, Y. M., Won, J. R. y Park, J. B.: *A new approach to economic load dispatch based on improved evolutionary programming*. Eng. Intell. Syst. Elect. Eng. Commu., 6:103–110, 1998.

[141] Parrales, K., Palau, C. y Delgado, B.: *Metaheurísticas de la Optimización*. http://www.fiec.espol.edu.ec/investigacion/topico/antcolonygrasp.pdf.

[142] Peer, E. S., van den Bergh, F. y Engelbrecht, A. P.: *Using Neighborhoods with the Guaranteed Convergence PSO*. En *Proceedings of the IEEE Swarm Intelligence Symposium*, páginas 235–242. IEEE Press, 2003.

[143] Peng, J., Chen, Y. y Eberhart, R. C.: *Battery Pack State of Charge Estimator Design using Computational Intelligence Approaches*. En *Proceeding of the Annual Battery Conference on Applications and Advances*, páginas 173–177, 2000.

[144] Perez, R. E. y Behdinan, K.: *Particle Swarm approach for Structural Design Optimization*. Computers and Structures, 85(19-20):1579–1588, 2007.

[145] Poli, R.: *An Analysis of Publications on Particle Swarm Optimisation Applications*. Journal of Artificial Evolution and Applications, 2008. ISSN: 1687-6229.

[146] Price, K.: *An Introduction to Differential Evolution*. En D. Corne, M. Dorigo y F. Glover (editor): *New Ideas in Optimization*, páginas 79–106. McGraw-Hill, 1999.

[147] Price, K. V.: *An Introduction to Differential Evolution*. En Corne, David, Dorigo, Marco y Glover, Fred (editores): *New Ideas in Optimization*, páginas 79–108. McGraw-Hill, London, UK, 1999.

[148] Ragsdell, K. y Phillips, D.: *Optimal Design of a Class of Welded Structures using Geometric Programming*. ASME Journal of Engineering for Industries, 98(3):1021–1025, 1976.

[149] Rajeev, S. y Krishnamoorthy, C. S.: *Genetic Algorithms-based methodologies for design optimization of trusses*. Journal of Structural Engineering, 123(3):350–358, 1997.

[150] Ranjithan, S. R., Chetan, S. K. y Dakshima, H. K.: *Constraint Method-Based Evolutionary Algorithm (CMEA) for Multiobjective Optimization*. En al., E. Zitzler et (editor): *First International Conference on Evolutionary Multi-Criterion Optimization*, páginas 299–313. Springer-Verlag, 2001. LNCS vol. 1993.

[151] Rao, S. S.: *Multiobjective Optimization in Structural Design with Uncertain Parameters and Stochastic Processes*. AIAA Journal, 22(11):1670–1678, 1984.

[152] Ratnaweera, A., Watson, H. y Halgamuge, S. K.: *Optimisation of Valve Timing Events of Internal Combustion Engines*. En *Proceedings of the IEEE Congress on Evolutionary Computation*, volumen 4, páginas 2411–2418, 2003.

[153] Rechenberg, I.: *Evolutionary Estrategy: Optimization of Technical Systems According to the Principles of Biological Evolution*. Frommann-Holzboog, Verlag, Stuttgart, 1973.

[154] Reklaitis, G. V., Ravindran, A. y Ragsdell, K. M.: *Engineering Optimization. Methods and Applications*. John Wiley and Sons, 1983.

[155] Reyes Sierra, M. y Coello Coello, C. A.: *Improving PSO-based Multi-Objective Optimization Using Crowding, Mutation and ϵ-Dominance*. En *Evolutionary Multicriterion Optimization, EMO 2005*, páginas 505–519, 2005. Lecture Notes in Computer Science, vol. 3410.

[156] Reyes-Sierra, M. y Coello Coello, C. A.: *Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art*. International Journal of Computational Intelligence Research, 2(3):287–308, 2006.

[157] Reynolds, R. G.: *Cultural Algorithms: Theory and Applications*. En Corne, D., Dorigo, M. y Glover, F. (editores): *New Ideas in Optimization*, página 367. McGraw-Hill, 1999.

[158] Rosenberg, R. S.: *Simulation of Genetic Populations with Biochemical Properties*. Tesis de Doctorado, University of Michigan, Ann Arbor, Michigan, USA, 1967.

[159] Runarsson, T. P.: *Approximate Evolution Strategy using Stochastic Ranking*. En *2006 IEEE Congress on Evolutionary Computation (CEC'2006)*, páginas 2760–2767, Vancouver, BC, Canada, July 2006. IEEE.

[160] Runarsson, T. P. y Yao, X.: *Stochastic Ranking for Constrained Evolutionary Optimization*. IEEE Transactions on Evolutionary Computation, 4(3):284–294, 2000.

[161] Salman, A., Ahmad, I. y Al-Madani, S.: *Particle Swarm Optimization for Task Assignment Problem*. Microprocessors and Microsystems, 26(8):363–371, 2002.

[162] Sanchez Carpena, G.: *Diseño y Evaluación de Algoritmos Evolutivos Multiobjetivo en Optimización y Modelización Difusa*. Tesis Doctoral, Noviembre 2002.

[163] Sandgren, E.: *Nonlinear Integer and Discrete Programming in Mechanical Design Optimization*. ASME Journal of Mechanical Design, 112:223–229, 1990.

[164] Santana-Quintero, L. V., Ramírez-Santiago, N., Coello Coello, C. A., Molina Luque, J. y García Hernández-Díaz, A.: *A New Proposal for Multiobjecive Optimization Using Particle Swarm Optimization and Rough Sets Theory*. En Runarsson, Thomas Philip, Beyer, Hans Georg, Burke, Edmund K., Guervós, Juan J. Merelo, Whitley, L. Darrell y Yao, Xin (editores): *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, volumen 4193 de *Lecture Notes in Computer Science*, páginas 483–492. Springer, 2006, ISBN 3-540-38990-3.

[165] Schaffer, J D.: *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. Tesis de Doctorado, Vanderbilt University, Nashville, Tennessee, 1984.

[166] Schutte, J. F. y Groenwold, A. A.: *Sizing Design of Truss Structures using Particle Swarms*. Structural and Multidisciplinary Optimization, 25(4):261–269, 2003.

[167] Schwefel, H.: *Numerical Optimization for Computer Models*. Wiley, Chichester, UK, 1981.

[168] Secrest, B. R.: *Travelling Salesman Problem for Surveillance Mission Planning using Particle Swarm Optimization*. Tesis de Licenciatura, School of Engineering and Management of the Air Force Institute of Technology, Air University, 2001.

[169] Shi, Y. y Eberhart, R.: *A modified Particle Swarm Optimizer*. En *Proceeding of the 1998 IEEE World Congress on Computational Intelligence*, páginas 69–73, Piscataway,NJ, 1998.

[170] Shi, Y. y Eberhart, R.: *Empirical Study of Particle Swarm Optimization*. En *Proceeding of the 1999 IEEE World Congress on Computational Intelligence*, páginas 1945–1950, Piscataway,NJ, 1999.

[171] Shi, Y. y Eberhart, R. C.: *Parameter Selection in Particle Swarm Optimization*. En *Proceeding of the Seventh Annual Conference on Evolutionary Programming*, páginas 591–600, 1998.

[172] Sinha, N., Chakrabarti, R. y Chattopadhyay, P. K.: *Evolutionary Programming Techniques for Economic Load Dispatch*. IEEE Transactions on Evolutionary Computation, 7(1):83–94, 2003.

[173] Smit, S. K. y Eiben, A. E.: *Comparing Parameter Tuning Methods for Evolutionary Algorithms*. En *IEEE Congress on Evolutionary Computation (CEC 2009)*, páginas 399–406, 2009.

[174] Song, Y. H. y Yu, I. K.: *Dynamic load dispatch with voltage security and environmental constraints*. Electric Power Systems Research, 43:53–60, 1997.

[175] Srinivasan, D., Loo, W. H. y Cheu, R. L.: *Traffic Incident Detection using Particle Swarm Optimization*. En *Proceedings of the IEEE Swarm Intelligence Symposium*, páginas 144–151, 2003.

[176] Sriyanyong, P., Song, Y. H. y Turner, P. J.: *Evolutionary Scheduling*, volumen 49 de *Studies in Computational Intelligence*, capítulo Particle Swarm Optimisation for Operational Planning: Unit Commitment and Economic Dispatch. Springer, 2007.

[177] Steuer, R. E.: *Multiple Criteria Optimization. Computation and Application*. Krieger, Malabar FL, 1986. 2da Edición.

[178] Thierauf, G. y Cai, J.: *Evolution Strategies-parallelization and Applications in Engineering Optimization*. En Topping, B. H. V. (editor): *Parallel and Distributed Processing for Computational Mechanics*, 1997.

[179] Tillett, J. C., Rao, R., Sahin, F. y Rao, T. M.: *Cluster-Head Identification in Ad Hoc Sensor Networks using Particle Swarm Optimization*. En *Proceedings of 2002 IEEE International Conference on Personal Wireless Communications*, páginas 201–205, 2002.

[180] Toscano, G. y Coello Coello, C.: *Using Clustering Techniques to Improve the Performance of a Multi-objective Particle Swarm Optimizer*. En *Genetic and Evolutionary Computation, GECCO 2004*, 2004. Vol. 3102 of LNCS, páginas 225-237.

[181] Toscano-Pulido, G., Coello Coello, C. A. y Santana-Quintero, L. V.: *EMOPSO: A Multi-Objective Particle Swarm Optimizer with Emphasis on Efficiency*. En Obayashi, Shigeru, Deb, Kalyanmoy, Poloni, Carlo, Hiroyasu, Tomoyuki y Murata, Tadahiko (editores): *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007*, páginas 272–285. Springer. Lecture Notes in Computer Science Vol. 4403, 2007.

[182] Trojanowski, K.: *Multi-Swarm That Learns*. En *Intelligent Information Systems 2008*, páginas 121–130, Vancouver, BC, Canada, 2008. IEEE.

[183] Tupia, M. y Mauricio, D.: *Un algoritmo voraz para resolver el problema de la programación de tareas dependientes en máquinas diferentes*. Revista de Investig. Sist. Inform. RISI 1 (1), páginas 9–18, 2004. ISSN: 1815-0268 (impreso).

[184] van der Merwe, D. W. y Engelbrecht, A. P.: *Data Clustering using Particle Swarm Optimization*. En *Proceedings of IEEE Congress on Evolutionary Computation 2003*, volumen 1, páginas 215–220, 2003.

[185] Van Veldhuizen, D. A.: *Multiobjective Evolutionary Algorithms: Classification, Analyses, and New Innovations*. Tesis de Doctorado, Department of Electrical and Computer Engineering. Graduate School og Engineering. Air Force Institute of Technology, Ohio, 1999.

[186] Van Veldhuizen, D. A. y Lamont, G. B.: *Multiobjective Evolutionary Algorithm Research: A History and Analysis*. Informe técnico, Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst. Technol, Wright-Patterson, AFB.OH, 1998. TR-98-03.

[187] Van Veldhuizen, D. A. y Lamont, G. B.: *Multiobjective Evolutionary Algorithms Test Suites*. En Carroll, J., Haddad, H., Oppenheim, D., Bryant, B. y Lamont, G. B. (editores): *Proceedings of the 1999 ACM Symposium in Applied Computing*, páginas 351–357, San Antonio, Texas, 1999.

[188] Van Veldhuizen, D. A. y Lamont, G. B.: *Multiobjective Evolutionary Algorithms: Analysing the State-of-the-Art*. Evolutionary Computation, 7(3):1–26, 2000.

[189] Velázquez Reyes, J.: *Propuesta de Evolución Diferencial para Optimización de Espacios Restringidos*. Tesis de Licenciatura, Departamento de Ingeniería Eléctrica,

Sección de Computación, Centro de Investigación y de Estudios Avanzados del IPN, México, 2006.

[190] Venter, G. y Sobieszczanski-Sobieski, J.: *Multidisciplinary Optimization of Transport Aircraft Wings using Particle Swarm Optimization*. En *Nineth AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization 2002*, 2002.

[191] Venter, G. y Sobieszczanski-Sobieski, J.: *Particle Swarm Optimization*. En *Journal for AIAA*, volumen 41, páginas 1583–1589, 2003.

[192] Victoire, T. A. A. y Jeyakumar, A. E.: *Reserve Constrained Dynamic Dispatch of Units with valve-point effects*. IEEE Trans. Power Syst., 20:1273–1282, 2005.

[193] von Neumann, J. y Morgenstern, O.: *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, New Jersey, 1947. Segunda Edición.

[194] Walters, D. C. y Sheble, G. B.: *Genetic algorithm solution of economic dispatch with valve point loading*. IEEE Trans. Power Syst., 8:1325–1332, 1993.

[195] Wierzbicki, A. P.: *On the Use of Penalty Functions in Multiobjetive Optimization*. En *Proceedings of the International Symposium on Operations Research*, 1978.

[196] Wolpert, D. H. y Macready, W. G.: *No Free Lunch Theorems for Optimization*. IEEE Transactions on Evolutionary Computation, 1(1):67–82, 1997.

[197] Wood, A. J. y Wollenberq, B. F.: *Power Generation, Operation and Control*. John Wiley, New York, 1984. 2da edición.

[198] Wu, S. y Chou, T.: *Genetic Algorithms for Nonlinear Mixed Discrete-integer Optimization Problems via Meta-genetic Parameter Optimization*. Engineering Optimization, 24:137–159, 1995.

[199] Yang, H. T., Yang, P. C. y Huang, C. L.: *Evolutionary Programming based Economic dispatch for units with non-smooth fuel cost functions*. IEEE Trans. Power Syst., 11:112–118, 1996.

[200] Yao, X. y Liu, Y.: *Fast Evolution Strategies*. En *Lecture Notes in Computer Science*, volumen 1213, páginas 149–161. Springer Berlin/Heidelberg, 1997.

[201] Zadeh, L. A.: *Optimality and Nonscalar-Valued Performance Criteria*. IEEE Transactions on Automatic Control, AC-8(1):59–60, 1963.

[202] Zhang, C., Shao, H. y Li, Y.: *Particle Swarm Optimization for Evolving Artificial Neural Network*. En *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, páginas 2487–2490, 2000.

[203] Zhang, C. y Wang, H.: *Mixed-discrete Nonlinear Optimization with Simulated Annealing*. Engineering Optimization, 21:277–291, 1993.

[204] Zhang, F. y Xue, D.: *Optimal Concurrent Design Based upon Distributed Product Development Life-Cycle Modelling*. Robotics and Computer-Integrated Manufacturing, 17(6):469–486, 2001.

[205] Zhao, S. Z., Liang, J. J., Suganthan, P. N. y Tasgetiren, M. F.: *Dynamic multi-swarm particle swarm optimizer with local search for Large Scale Global Optimization*. En *2008 IEEE Congress on Evolutionary Computation (CEC'2008)*, páginas 3845–3852, Hong Kong, 2008. IEEE Press.

[206] Zheng, Y. L., Ma, L. H., Zhang, L. Y. y Qian, J. X.: *Empirical Study of Particle Swarm Optimizer with an increasing inertia weight*. En *Proceeding of the 2003 IEEE World Congress on Computational Intelligence*, páginas 221–226, Piscataway,NJ, 2003.

[207] Zitzler, E., Deb, K. y Thiele, L.: *Comparison of the Multiobjective Evolutionary Algorithms: Empirical Results*. Evolutionary Computation, 8(2):173–195, 2000.