Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Optimización multi-objetivo usando un algoritmo compacto de cúmulos de partículas

Tesis que presenta

Jorge Jiménez Montiel

Para obtener el grado de

Maestro en Ciencias en Computación

Director de la tesis:

Dr. Carlos Artemio Coello Coello

Ciudad de México Octubre, 2017

Agradecimientos

Al CINVESTAV y en específico al departamento de computación, por darme la oportunidad de realizar mis estudios en esta gran institución. También quiero agradecer al CONACyT por la beca económica que me otorgó y que permitió que pudiera realizar y concluir mis estudios de maestría. El tema que dio pie a este trabajo de tesis fue derivado del proyecto CONACyT titulado "Nuevos Paradigmas Algorítmicos en Optimización Evolutiva Multi-Objetivo"(Ref. 221551), cuyo responsable técnico es el Dr. Carlos A. Coello Coello.

Quiero agradecer de forma especial al Dr. Carlos Coello Coello, por la oportunidad de trabajar bajo su supervisión y por sus valiosos comentarios y observaciones.

También quiero agradecer a mis sinodales, el Dr. Luis Gerardo de la Fraga y el Dr. Esteban Tlelo Cuautle por su tiempo dedicado y por sus observaciones que me ayudaron a mejorar el contenido de esta tesis.

Finalmente, quiero agradecer a mi familia y amigos, por su apoyo incondicional y por los valiosos consejos que me brindaron durante este tiempo.

Resumen

Gran cantidad de problemas en el mundo real requieren la optimización de varios objetivos (normalmente en conflicto entre sí) de manera simultánea. Estos son los denominados problemas de optimización multi-objetivo (POM). La solución de estos problemas, consiste en un conjunto de soluciones que representa los mejores compromisos entre los objetivos del problema. El conjunto de todas las soluciones para un problema multi-objetivo se denomina *onjunto óptimo de Pareto* y la imagen de dicho conjunto sobre el espacio de objetivos se denomina *frente de Pareto*

Han surgido varios métodos de programación matemática para resolver este tipo de problemas. Sin embargo, estos métodos requieren información adicional del problema o son suceptibles a la forma y continuidad del frente de Pareto.

A lo largo de los años se han propuesto distintas metaheurísticas inspiradas tanto en el proceso evolutivo como en el comportamiento social de ciertas especies de animales. Estas metaheurísticas han sido utilizadas ampliamente para la solución de problemas multi-objetivo, ya que éstas no tienen las limitantes de los métodos de programación matemática.

Sin embargo, la gran mayoría de las metaheurísticas no son aptas para aplicaciones en donde los recursos son limitados, ya sea porque la cantidad de memoria disponible es restringida o el costo computacional para evaluar a la función es muy alto.

En esta tesis se propone una nueva metaheurística compacta para problemas multi-objetivo basada en *optimización mediante cúmulos de partículas* (PSO, por sus siglas en inglés), la cual tiene como finalidad resolver problemas de optimización multi-objetivo reduciendo a la vez la cantidad de memoria requerida.

Abstract

A large number of real-world problems require the simultaneous optimization of several objectives (which are normally in conflict with each other). These are the so-called multi-objective optimization problems (MOPs). The solution of these problems consists of a set of solutions representing the best possible trade-off among the problem's objectives. The set containing all the possible solutions to a multi-objective problem is called Pareto Optimal Set and its image (i.e., objective function values) is denominated Pareto front.

A number of mathematical programming techniques have been proposed to solve multi-objective problems. However, such methods require additional information about the problem or are very susceptible to the shape and continuity of the Pareto front.

Throughout the years, several bio-inspired metaheuristics have been proposed. Such approaches include metaheuristics inspired on the evolutionary process and the social behavior of certain animal species. These metaheuristics have been widely used to solve multi-objective problems, because they do not present the limitations of mathematical programming techniques that were previously indicated.

Nevertheless, most of the current metaheuristics are not appropriate for applications having limited resources, either due to limited available memory or to computationally expensive objective function evaluations.

In this thesis, we propose a new compact metaheuristics for solving multi-objective optimization problems, which is based on a particle swarm optimizer. Its main goal is to solve multi-objective optimization problems using a very limited amount of memory.

Índice general

1 Introducción **1**

1.1. Motivación 1

1.2. Planteamiento del problema 2

1.3. Objetivos generales y específicos 3

1.4. Organización de la tesis 3

2 Conceptos de optimización multi-objetivo **5**

2.1. Optimización multi-objetivo 5

2.2. Optimalidad de Pareto 6

2.3. Vectores de referencia 8

2.3.1. Vector objetivo ideal 8

2.3.2. Vector objetivo utópico 9

2.3.3. Vector objetivo de nadir 10

2.4. Técnicas tradicionales de optimización multi-objetivo 11

2.4.1. Métodos sin preferencias 11

2.4.2. Métodos a posteriori 12

2.4.3. Métodos a priori 14

2.4.4. Métodos interactivos 15

2.5. Limitantes de las técnicas tradicionales de optimización multi-objetivo . . 16

3 Cómputo evolutivo **17**

3.1. Conceptos preliminares 18

3.1.1. Principales paradigmas 21

3.1.2. Algoritmo genético . . . 21
3.1.3. Programación evolutiva . . . 22
3.1.4. Estrategias evolutivas . . . 23
3.2. Algoritmos evolutivos multi-objetivo . . . 24
3.2.1. Elitismo . . . 24
3.3. Optimización mediante cúmulos de partículas . . . 25
3.3.1. PSO visto en términos sociales y cognitivos . . . 26
3.3.2. Descripción de la metaheurística . . . 27
3.3.3. PSO visto como un algoritmo evolutivo . . . 29
3.3.4. Topologías . . . 31
3.3.5. Optimización mediante cúmulos de partículas para problemas de optimización multiobjetivo . . . 33
3.4. Estado del arte en AEMOs y MOPSOs . . . 35

4 Metaheurísticas compactas 41
4.1. Algoritmo genético compacto . . . 41
4.1.1. Selección y cruza . . . 42
4.1.2. Simulación de una presión de selección mayor . . . 43
4.2. Algoritmo genético compacto con elitismo persistente y no persistente . . . 44
4.3. Real-Valued Compact Genetic Algorithm . . . 49
4.3.1. Generación de nuevos individuos . . . 50
4.3.2. Actualización del vector de probabilidades . . . 51
4.4. Optimizador mediante cúmulos de partículas compacto . . . 53
4.5. Evolución diferencial compacta . . . 55
4.6. Evolución diferencial compacta multi-objetivo . . . 56
4.6.1. Algoritmo para almacenar soluciones no dominadas . . . 58

5 Optimización multi-objetivo usando un algoritmo compacto de cúmulos de partículas 61
5.1. Importancia de la población . . . 62
5.2. Construcción del algoritmo mono-objetivo . . . 63

5.2.1. Representación de la población 63
5.2.2. Construcción de la metaheurística 65
5.2.3. Actualización del vector de probabilidades 66
5.3. *compactMOPSO* 72
5.3.1. Archivo externo 72
5.3.2. Algoritmo general del *compactMOPSO* 74

6 Estudio experimental **77**
6.1. Problemas de prueba 77
6.2. Parámetros utilizados 78
6.3. Descripción de la metodología utilizada 78
6.4. Comparación de desempeño 80
6.4.1. Comparación de MOPSOs mediante HV 81
6.4.2. Comparación de AEMOs compactos mediante HV 81
6.4.3. Comparación de MOPSOs mediante SP 83
6.4.4. Comparación de AEMOs compactos mediante SP 84
6.4.5. Comparación de MOPSOs mediante IGD+ 86
6.4.6. Comparación de AEMOs compactos mediante IGD+ 87
6.5. Resumen 90

7 Conclusiones y trabajo futuro **93**

A Problemas de prueba **95**
A.1. Conjunto de problemas de prueba Zitzler-Deb-Thiele 95
A.2. Problemas de prueba estándar 102

B Indicadores de desempeño para optimización multi-objetivo **113**
B.1. Hipervolumen 114
B.2. Espaciado 114
B.3. IGD+ 115

C Resultados completos **117**

C.1. Gráficas frentes de Pareto 118

C.2. Resultados gráficos de los indicadores 132

C.2.1. Optimizadores multi-objetivo basados en cúmulos de partículas (MOPSOs, por sus siglas en inglés) 132

C.2.2. AEMOs 147

Bibliografía **162**

Capítulo

1

Introducción

1.1. Motivación

Los problemas multi-objetivo se presentan de manera natural en diversas disciplinas de la ciencia e ingeniería. El uso de metaheurísticas bio-inspiradas para resolver este tipo de problemas se debe principalmente a su naturaleza poblacional, la cual les permite obtener múltiples elementos del frente de Pareto durante una sola ejecución.

Dentro de las metaheurísticas bio-inspiradas existe una clase conocida como inteligencia de enjambres que se caracteriza por reproducir el comportamiento social presente en diversas especies de animales. Estas metaheurísticas están típicamente formadas por una población de agentes o individuos simples que interactúan entre sí con el objetivo de resolver un problema.

En este trabajo de tesis nos enfocaremos específicamente en la *optimización mediante cúmulos de partículas* (PSO por sus siglas en inglés), propuesta por Eberhart y Kennedy en 1995 [30], la cual es una metaheurística perteneciente a la clase de inteligencia de enjambres. PSO es un procedimiento de búsqueda poblacional, donde cada individuo es abstraído como una partícula que se encuentra en un espacio de búsqueda multi-dimensional. Las mejores posiciones encontradas por las partículas y sus vecinos determinan la trayectoria de las partículas en las siguientes iteraciones logrando así un balance entre la exploración y la explotación combinando mecanismos de búsqueda local y global [40].

Entre las principales ventajas del PSO se encuentran las siguientes: requiere una cantidad pequeña de parámetros respecto a los algoritmos evolutivos, es fácil de implementar y usar y suele converger más rápido que los algoritmos evolutivos en algunos problemas de prueba [47, 32, 7].

1.2. Planteamiento del problema

En esta tesis se propone una nueva metaheurística compacta para problemas multi-objetivo basada en *PSO*.

Esto representa un reto debido a que la capacidad de búsqueda de un PSO reside principalmente en la interacción existente en su población. Al restringir el número de partículas a una, se pierde tanto la diversidad como la capacidad de búsqueda en problemas de alta dimensionalidad [43]. Además, al no tener una población, dejan de existir las topologías para la comunicación entre partículas, lo cual puede aumentar las probabilidades de quedar atrapado en un óptimo local en problemas multi-modales[1] [33, 31]. Así mismo, al contar solo con un individuo, no existe diferencia entre el concepto de mejor posición local (*lbest*) y mejor posición global (*gbest*); es decir, la mejor posición de la partícula también es la mejor posición de la población. Por lo tanto, existe un desequilibrio entre la búsqueda local y global.

Finalmente, existen ciertas cuestiones que se deben responder al momento de adaptar un PSO para resolver problemas multi-objetivo:

- ¿Cómo decidir cuál partícula utilizar como el líder global?
- ¿Cómo conservar todas las soluciones no dominadas encontradas durante la búsqueda?
- ¿Cómo mantener la diversidad de forma que se evite la convergencia a una única solución?

[1] Un problema multi-modal, es aquel que posee numerosos óptimos ya sea locales o globales.

1.3. Objetivos generales y específicos

Objetivo General

Diseñar e implementar una metaheurística compacta basada en PSO para problemas de optimización multi-objetivo sin restricciones.

Objetivos específicos

- Diseñar un PSO compacto mono-objetivo inspirado en el diseño del algoritmo genético compacto (CGA, por sus siglas en inglés).
- Extender el PSO compacto a problemas multi-objetivo utilizando un enfoque basado en descomposición.
- Comparar el desempeño del PSO compacto multi-objetivo respecto a metaheurísticas multi-objetivo sin restricciones del estado del arte, utilizando problemas de prueba e indicadores de desempeño estándar de la literatura especializada.

1.4. Organización de la tesis

Esta tesis comprende siete capítulos y tres apéndices y se encuentra organizada de la siguiente forma:

El capítulo dos tiene el objetivo de presentar los conceptos básicos de la optimización multi-objetivo. Dichos conceptos pretenden servir como guía al lector en caso de no estar familiarizado con el tema. Se describe el método de las métricas ponderadas que posteriormente es utilizado para resolver problemas de optimización multi-objetivo.

El capítulo tres describe de forma breve los conceptos básicos del cómputo evolutivo, iniciando con los principales paradigmas. Posteriormente, se presenta al optimizador mediante cúmulo de partículas que se usará como motor de búsqueda en esta tesis. Primero se proporciona su descripción para problemas mono-objetivo, describiéndose algunos algoritmos del estado del arte para resolver problemas multi-objetivo.

El cuarto capítulo presenta a las metaheurísticas compactas, iniciando con la primera metaheurística compacta que se propuso (el algoritmo genético compacto), indicando su motivación principal. Posteriormente, se discute el estado del arte en torno a las metaheurísticas compactas. En la parte final del capítulo se describe la evolución diferencial compacta para problemas multi-objetivo, que es la única metaheurística compacta de la cúal se tiene conocimiento hasta la fecha, que tiene como propósito resolver problemas de optimización multi-objetivo.

El quinto capítulo tiene como objetivo el describir de forma detallada la propuesta para resolver problemas multi-objetivo empleando una representación estadística de la población.

El sexto capítulo muestra los resultados obtenidos al comparar la propuesta con dos optimizadores multi-objetivo basados en cúmulos de partículas del estado del arte y con un algoritmo evolutivo multi-objetivo compacto del estado del arte.

En el séptimo y último capítulo se presentan las conclusiones de este trabajo de tesis y se discuten algunos aspectos y trabajos posibles a futuro.

Capítulo

2

Conceptos de optimización multi-objetivo

2.1. Optimización multi-objetivo

El problema de interés para esta tesis es el problema de optimización multi-objetivo sin restricciones, el cual consiste en minimizar (o maximizar) m funciones objetivo de manera simultánea que usualmente se encuentran en conflicto entre sí. Formalmente, este problema se define como:

$$\text{minimizar} \quad F : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^m \tag{2.1}$$

donde el dominio de la función (Ω) corresponde a la región factible del problema y satisface las restricciones implícitas del problema multi-objetivo[1], Ω es conocido como el espacio de variables de decisión y $x \in \Omega$ como variable de decisión. La imagen de la función ($\mathbb{R}^m$) es conocida como el espacio objetivo (véase figura 2-1). La función vectorial $F : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^m$ es denominada función objetivo y cada componente de la función objetivo se define como $f_i : \mathbb{R}^n \rightarrow \mathbb{R}, \ \forall i = \{1, ..., m\}$ para $m \geq 2$, $F(x) = (f_1(x), ..., f_m(x))$, $x \in \Omega$.
Se supondrá a lo largo de este trabajo, sin pérdida de generalidad, que todas las funciones objetivo se minimizan; dado que maximizar f equivale a minimizar $-f$.

[1]Una restricción impícita corresponde al rango en el que una variable de decisión es válida. Dados $a, b \in \mathbb{R}, \ a \leq b$, se dice que x tiene una restricción implícita en a y b: $a \leq x \leq b$

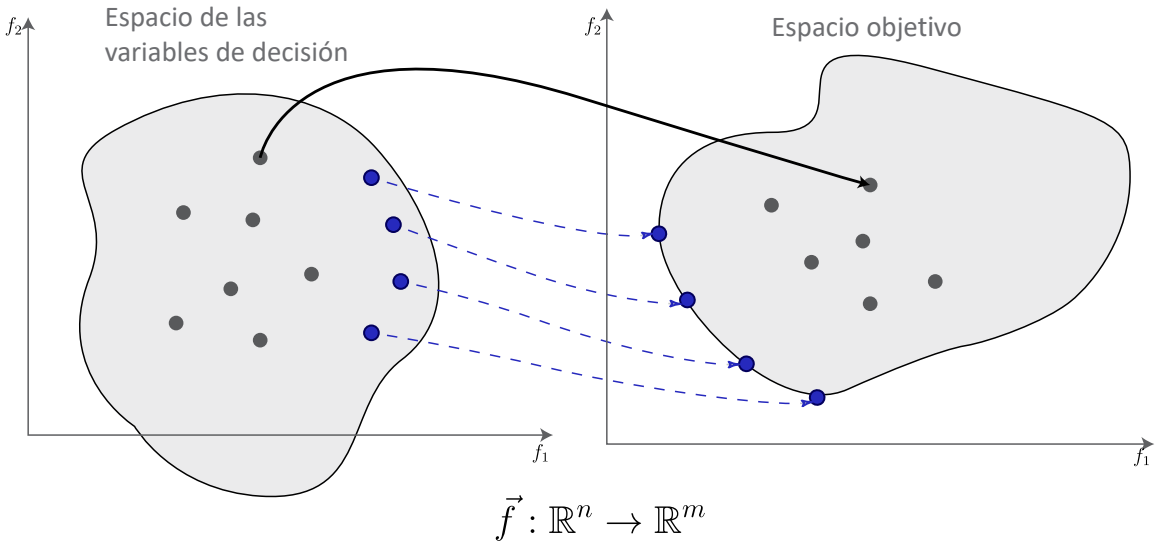


Figura 2-1: Espacios de búsqueda en un problema de optimización con 2 objetivos

2.2. Optimalidad de Pareto

En la sección anterior se definió el problema de optimización multi-objetivo como minimizar $F : \mathbb{R}^n \to \mathbb{R}^m$ para $m \geq 2$.

Cuando $m = 1$ se tiene el problema: minimizar $F : \mathbb{R}^n \to \mathbb{R}$ denominado problema de optimización mono-objetivo. Para este caso, hallar el mínimo consiste en encontrar un vector de decisión $x \in \mathbb{R},\ x > -\infty$ tal que no exista otro vector de decisión $y \in \mathbb{R}$ que sea menor que él; es decir, $f(x) \leq f(y)\ \forall y \in \mathbb{R}$.

Sin embargo, no existe un orden natural para el conjunto $\mathbb{R}^m$ por lo que no es posible establecer el mínimo entre dos soluciones de dicho conjunto. En esta sección se presenta la dominancia de Pareto a partir de la cual es posible establecer el concepto de optimalidad de Pareto.

Definición 2.2.1. ***Dominancia de Pareto*** *Se dice que una solución $u = (u_1, ..., u_m)$ domina a otra solución $v = (v_1, ..., v_m)$ denotado como $u \preceq v$ si y sólo si*

$$\forall i \in \{1, ..., m\},\ \ u_i \leq v_i\ \ \wedge\ \ \exists j \in \{1, ..., m\} : u_j < v_j$$

Definición 2.2.2. ***Dominancia estricta de Pareto*** *Se dice que una solución $u = (u_1, ..., u_m)$*

domina estrictamente a otra solución $v = (v_1, ..., v_m)$ *denotado como* $u \prec v$ *si y sólo si*

$$\forall i \in \{1, ..., m\}, \;\; u_i < v_i$$

Definición 2.2.3. ***Optimalidad de Pareto*** *Se dice que una solución* $u = (u_1, ..., u_m)$ *es Pareto-óptima con respecto a* Ω *si y sólo si*

$$\nexists v \in \Omega : v \preceq u$$

Definición 2.2.4. ***Optimalidad débil de Pareto*** *Se dice que una solución* $u = (u_1, ..., u_m)$ *es Pareto-óptima débil con respecto a* Ω *si y sólo si*

$$\nexists v \in \Omega : v \prec u$$

Definición 2.2.5. ***Conjunto de óptimos de Pareto*** *Dado un problema de optimización multi-objetivo,* $F(x)$*, el conjunto de óptimos de Pareto, denotado por* $\mathcal{P}^*$ *es definido como*

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

Definición 2.2.6. ***Frente óptimo de Pareto*** *Dado un problema de optimización multi-objetivo,* $F(x)$*, el frente óptimo de Pareto, denotado por* $\mathcal{PF}^*$ *es definido como*

$$\mathcal{PF}^* = \{F(x) \mid x \in \mathcal{P}^*\}$$

La figura 2-2 ejemplifica el concepto de frente óptimo de Pareto.

Definición 2.2.7. ***Función convexa*** *Una función* $f_i : \mathbb{R}^n \to \mathbb{R}$ *es convexa si* $\forall x_1, x_2 \in \mathbb{R}^n$ *se cumple*

$$f_i(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f_i(x_1) + (1 - \lambda) f_i(x_2) \, \forall \lambda \in [0, 1]$$

En otras palabras, f_i es una función convexa si para cualquier punto situado entre x_1 y x_2 **(**$\lambda x_1 + (1 - \lambda)x_2$**)**, su valor, al ser evaluado en la función, es menor o igual al valor que

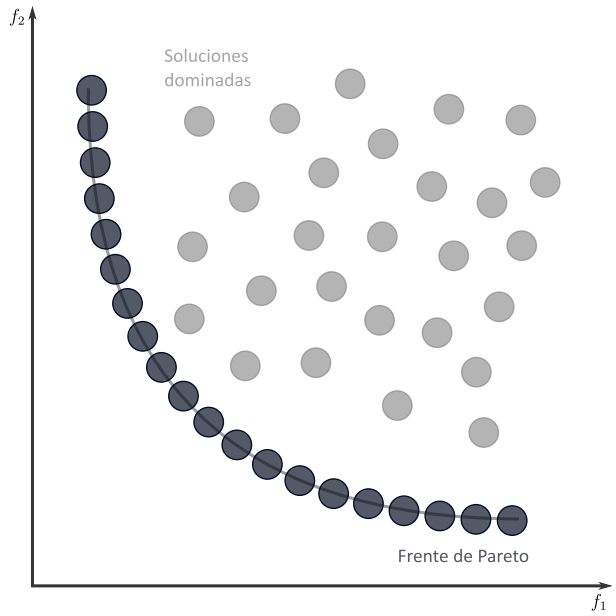


Figura 2-2: Ejemplo de un frente de Pareto y de soluciones dominadas en un espacio de dos dimensiones.

toma dicho punto en el segmento de recta que conecta a x_1 y x_2. Esto se muestra de forma gŕafica en la figura 2-3.

Definición 2.2.8. ***Conjunto convexo*** *Un conjunto $S \subset \mathbb{R}^n$ es convexo si $x_1, x_2 \in S$ implica que $\lambda x_1 + (1-\lambda)x_2 \in S$ para toda $\lambda \in [0,1]$.*

Definición 2.2.9. *Un problema de optimización multi-objetivo es convexo si todas las funciones objetivo y la región factible son convexos.*

2.3. Vectores de referencia

A continuación se presentan algunos vectores que son utilizados como soluciones de referencia y que pueden ayudar a ciertos métodos para resolver problemas multi-objetivo a acercarse al frente de Pareto.

2.3.1. Vector objetivo ideal

El vector objetivo ideal denotado por z^*, representa los límites inferiores de cada objetivo en el espacio de búsqueda factible. Está constituido por los valores óptimos individuales

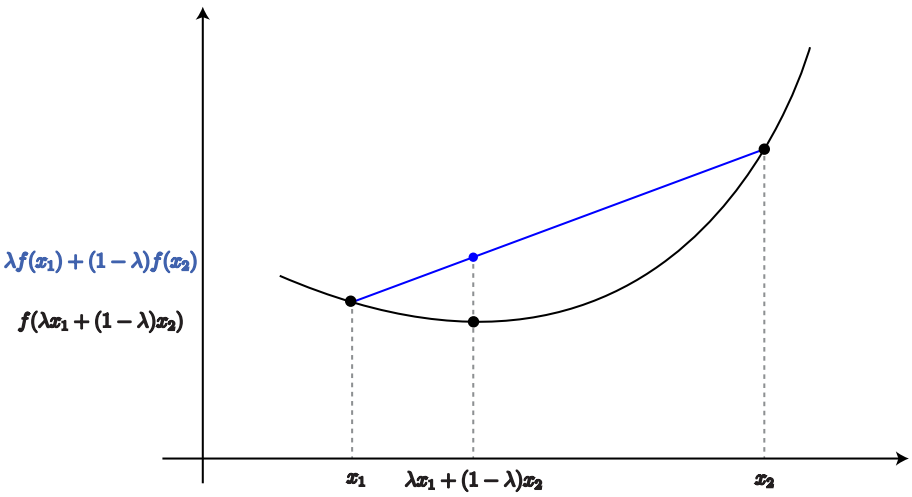


Figura 2-3: Ejemplo de una función convexa en el intervalo x_1 a x_2

para cada función objetivo

$$\begin{aligned} z_i^* &= \min f_i(x), \quad x \in \Omega \\ z^* &= (z_1^*, z_2^*, ..., z_m^*)^T \end{aligned} \tag{2.2}$$

En general, el vector objetivo ideal corresponde a una solución no factible en el espacio objetivo. La única forma en que un vector objetivo ideal sea factible es que no exista conflicto entre los objetivos [15]. Usualmente es utilizado como un punto de referencia, dado que las soluciones factibles más cercanas a éste representan el mejor compromiso entre los objetivos.

2.3.2. Vector objetivo utópico

Algunas veces se desea un vector objetivo que domine estrictamente a todas las soluciones del conjunto de óptimos de Pareto. A este vector se le denomina vector objetivo utópico, se le denota mediante z^{**} y se define de la siguiente forma

$$z_i^{**} = z_i^* - \epsilon_i, \quad \forall\, i = \{1, ..., m\}, \quad \epsilon_i \geq 0 \tag{2.3}$$

Un caso en donde el vector objetivo utópico es utilizado es el problema de Tchebycheff ponderado que se discutirá en la sección 2.4.2.

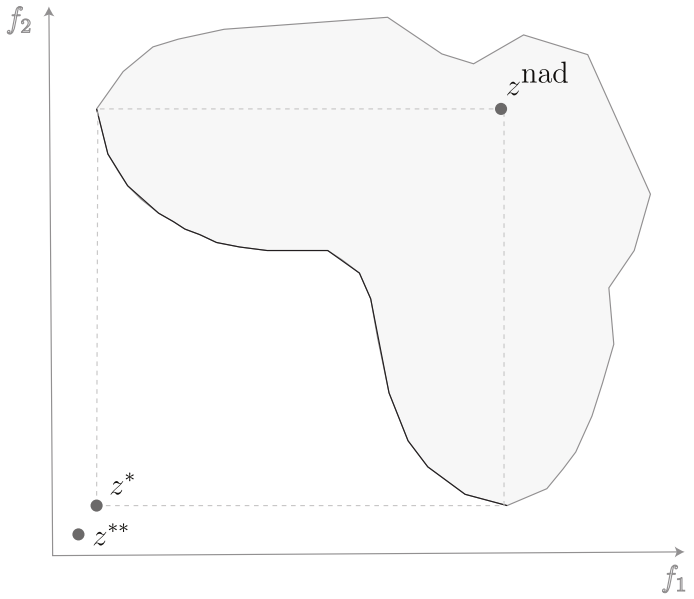


Figura 2-4: Vectores de referencia: vector ideal (z^*), vector de Nadir (z^{nad}) y vector utópico (z^{**})

2.3.3. Vector objetivo de nadir

El vector objetivo de Nadir denotado por z^{nad}, representa los límites superiores de cada objetivo en el conjunto de óptimos de Pareto ($\mathcal{P}^*$)

$$\begin{aligned} z_i^{nad} &= \max f_i(x), \;\; x \in \mathcal{P}^* \\ z^{nad} &= \left(z_1^{nad}, z_2^{nad}, ..., z_m^{nad}\right)^T \end{aligned} \tag{2.4}$$

Dependiendo de la continuidad y forma del frente de Pareto, el vector objetivo de Nadir puede o no ser una solución factible. En muchas ocasiones, es recomendable normalizar las funciones objetivo de tal forma que los valores objetivo tengan aproximadamente la misma magnitud. Si tanto el vector objetivo ideal como el vector objetivo de Nadir son conocidos, es posible reemplazar la función $f_i(x)$ por la función

$$\frac{f_i(x) - z_i^*}{z_i^{nad} - z_i^*} \tag{2.5}$$

2.4. Técnicas tradicionales de optimización multi-objetivo

Un tomador de decisiones es el que puede expresar una relación de preferencia sobre un conjunto de soluciones.

En el proceso de resolver un problema de optimización multi-objetivo, se requiere usualmente de alguna información que exprese las preferencias del tomador de decisiones. Con base en la información requerida, Hwang y Masud [3] propusieron una clasificación de los métodos para resolver problemas multi-objetivo. A continuación se da una descripción breve de cada una de las clases que componen dicha clasificación.

2.4.1. Métodos sin preferencias

Los métodos que sigan este enfoque no necesitan información sobre la preferencia del tomador de decisiones una vez que tanto las restricciones como los objetivos del problema han sido definidos. Por tanto, este enfoque requiere que el tomador de decisiones sea capaz de aceptar la solución obtenida por este método.

Entre los métodos más relevantes en esta clase se encuentra el método del criterio global, el cual se describe a continuación.

Método del criterio global

También se le conoce como *programación de compromisos*. En este método, la distancia existente entre algún punto de referencia y la región objetivo factible es minimizada. La forma más común para esta función es:

$$f(x) = \sum_{i=1}^{m} \left(\frac{z_i^* - f_i(x)}{z_i^*} \right)^p \tag{2.6}$$

Otro tipo de métricas que miden la cercanía hacia al punto de referencia son las métricas L_p. En este caso, el vector objetivo ideal es usado como punto de referencia:

$$\begin{aligned} &\text{minimizar} \quad \left(\sum_{i=1}^{m} |f_i(x) - z_i^*|^p \right)^{1/p} \\ &\text{sujeto a} \;\; x \in \Omega \end{aligned} \tag{2.7}$$

Tomando en cuenta que no debería existir una solución que domine al vector objetivo ideal, no debería ser necesario utilizar el valor absoluto al realizar la diferencia. Sin embargo, en muchas ocasiones no se conoce al vector objetivo ideal global en un inicio, por lo que el método podría no funcionar como se espera.

Cuando $p = \infty$, la métrica es también llamada *Métrica de Tchebycheff* y el problema de Tchebycheff es de la forma

$$\begin{aligned} &\text{minimizar} \quad \max_{i=1,\ldots,m} \; |f_i(x) - z_i^*| \\ &\text{sujeto a} \;\; x \in \Omega \end{aligned} \tag{2.8}$$

Cabe destacar que la solución tanto de un problema de Thebycheff como de un problema L_p es un óptimo débil de Pareto. La demostración a esta afirmación se puede consultar en [35].

2.4.2. Métodos a posteriori

Los métodos de esta clase determinan un subconjunto del conjunto de óptimos de Pareto para un problema multi-objetivo. De este subconjunto de soluciones, el tomador de decisiones elige la solución más satisfactoria de acuerdo a cierto criterio a su elección no indicado previamente o no cuantificable. El principal inconveniente de los métodos que siguen este enfoque es que el proceso de generación es usualmente costoso en términos computacionales.

Entre los métodos más relevantes en esta clase se encuentran el método de ponderación y el método de las métricas ponderadas, los cuales se definen a continuación.

Método de ponderación

La idea es asociar cada función objetivo con un coeficiente de ponderación y minimizar la suma ponderada de los objetivos. De esta forma, las funciones multi-objetivo son reducidas a una función mono-objetivo.

Se asume que los coeficientes w_i son números reales tales que $w_i \leq 0 \ \ \forall i = 1, ..., m$. También se supone que los coeficientes de ponderación están normalizados, es decir, $\sum_{i=1}^{m} w_i = 1$. Dicho de otra forma, el problema de optimización multi-objetivo es transformado en el siguiente problema, que es denominado *problema de ponderación*

$$\begin{aligned} &\text{minimizar} \ \sum_{i=1}^{m} w_i f_i(x) \\ &\text{sujeto a} \ \ x \in \Omega, \end{aligned} \tag{2.9}$$

donde $w_i \leq 0 \ \ \forall i = 1, ..., m$ y $\sum_{i=1}^{m} w_i = 1$

La solución de un problema de ponderación es siempre un óptimo de Pareto si todos los coeficientes de ponderación son positivos o si la solución es única [35]. La princpial debilidad del método de ponderaciones es que no todas las soluciones óptimas de Pareto pueden ser encontradas a menos que el problema sea convexo.

Teorema 2.4.1. *Supongamos un problema de optimización multi-objetivo convexo. Si $x^* \in \Omega$ es un óptimo de Pareto, entonces existe un vector de ponderación $\boldsymbol{w}$ ($w_i \leq 0 \ \ \forall i = 1, ..., m$ y $\sum_{i=1}^{m} w_i = 1$) tal que x^* es una solución de un problema de ponderación.*

Este teorema obtenido de [35] permite concluir que cualquier solución óptima de Pareto puede ser obtenida mediante el método de ponderaciones. Sin embargo, el utilizar un conjunto de vectores de ponderación uniformemente distribuidos no necesariamente produce una distribución uniforme de soluciones, incluso si el problema es convexo [8].

Método de las métricas ponderadas

El método de las métricas ponderadas es una extensión ponderada del método del criterio global. Es posible obtener distintas soluciones alterando los coeficientes de ponderación w_i ($w_i \leq 0 \ \forall i = 1, ..., m$ y $\sum_{i=1}^{m} w_i = 1$) en las métricas L_p y de Tchebycheff. El problema

L_p ponderado para minimizar la distancia al punto de referencia ahora tiene la forma

$$\begin{aligned} &\text{minimizar} \quad \left(\sum_{i=1}^{m} w_i |f_i(x) - z_i^*|^p \right)^{1/p} \\ &\text{sujeto a} \quad x \in \Omega \end{aligned} \tag{2.10}$$

y el problema de Tchebycheff es de la forma

$$\begin{aligned} &\text{minimizar} \quad \max_{i=1,\ldots,m} \; w_i |f_i(x) - z_i^*| \\ &\text{sujeto a} \quad x \in \Omega \end{aligned} \tag{2.11}$$

La solución a un problema de Tchebycheff ponderado es un óptimo débil de Pareto si todos los coeficientes de ponderación son positivos. Al reemplazar al vector objetivo ideal con el vector utópico, entonces es posible obtener cualquier solución óptima de Pareto [35]. Finalmente, cabe destacar que algunas de las soluciones obtenidas por esta métrica son débilmente dominadas.

2.4.3. Métodos a priori

En el caso de los métodos a priori, el tomador de decisiones deberá especificar sus preferencias, deseos y opiniones antes de iniciar el proceso de solución para un problema de optimización multi-objetivo. La dificultad presente en los métodos que siguen este enfoque es que el tomador de decisiones puede tener una expectativa muy irrealista de lo que se pueda obtener.

Entre los métodos más relevantes de esta clase se encuentran el método de la función de utilidad y el ordenamiento lexicográfico, los cuales se describen a continuación.

Método de la función de utilidad

En el método de la función de utilidad, el tomador de decisiones deberá de ser capaz de dar de forma explícita y precisa una función matemática de la forma $U : \mathbb{R}^m \to \mathbb{R}$ que represente sus preferencias globales. Esta función deberá proveer un orden total sobre el

espacio objetivo. Entonces, el problema de la función de utilidad se define somo;

$$\begin{aligned} &\text{maximizar } U(f(x)) \\ &\text{sujeto a } x \in \Omega \end{aligned} \tag{2.12}$$

Aunque el método de la función de utilidad parece ser muy simple, su dificultad radica en especificar la expresión matemática para dicha función.

Ordenamiento lexicográfico

En el ordenamiento lexicográfico el tomador de decisiones deberá de ordenar las funciones objetivo de acuerdo a su importancia absoluta. Después de ordenar los objetivos, la función objetivo más importante es minimizada sujeta a las restricciones originales. Si el problema tiene una única solución, ésta es la solución final del problema multi-objetivo. De lo contrario, la segunda función más importante es minimizada. Además de considerar las restricciones originales de la segunda función ahora se debe garantizar que la función objetivo más importante conserve su valor óptimo. Si el problema tiene una única solución, ésta es la solución final del problema multi-objetivo. De lo contrario se repite el procedimiento mencionado. Se puede escribir el problema lexicográfico como

$$\begin{aligned} &\text{minimizar lex } f_1(x), f_2(x), ..., f_m(x) \\ &\text{sujeto a } x \in \Omega \end{aligned} \tag{2.13}$$

2.4.4. Métodos interactivos

Esta clase de métodos se basa en la definición progresiva de las preferencias del tomador de decisiones junto con la exploración del espacio objetivo. El tomador de decisiones trabaja junto con un analista o un programa de computadora interactivo. Se puede decir que el analista trata de determinar la estructura de las preferencias del tomador de decisiones de manera interactiva. En general, un método interactivo se puede descomponer en los siguientes pasos:

(a) Encontrar una solución factible inicial

(b) Interactuar con el tomador de decisiones

(c) Obtener una nueva solución (o un conjunto de soluciones). Si la nueva solución o una de las soluciones anteriores es aceptable, entonces termina la búsqueda de la solución. De lo contrario, regresa al paso b.

Entre los métodos más importantes en esta clase se encuentra el método de satisfación de metas que es discutido a continuación.

Método de satisfacción de metas

En este método, el tomador de decisiones especifica un conjunto inicial de niveles de satisfacción de metas L_j, $j = 1, 2, ..., m$ que deben ser factibles y posteriormente se identifica la función objetivo con el nivel de meta menos satisfactorio LS. A continuación, se optimiza dicha función (LS) sujeta a sus restricciones originales y a restricciones adicionales conformadas por el resto de las funciones objetivo. El analista, que trabaja con el tomador de decisiones, puede ajustar una o más metas hasta que una solución más favorable sea alcanzada.

2.5. Limitantes de las técnicas tradicionales de optimización multi-objetivo

Las técnicas antes descritas presentan varias limitantes. Por ejemplo, varias de ellas requieren que las funciones objetivo y las restricciones sean diferenciables. Otras no son adecuadas cuando el frente de Pareto está desconectado o es cóncavo. Adicionalmente, la mayoría de ellas generan una solución por ejecución y no hay garantía de que al cambiar el punto inicial de búsqueda, la solución final sea distinta de una generada anteriormente. Estas limitantes han motivado el uso de metaheurísticas para resolver problemas multi-objetivo, destacando principalmente los algoritmos evolutivos, que se han vuelto una opción muy popular en años recientes, como se describe en el siguiente capítulo.

Capítulo

3

Cómputo evolutivo

A pesar de poseer una gran velocidad de convergencia debido al uso de gradientes para determinar la dirección de búsqueda, la mayoría de los métodos clásicos tienen dificultades para resolver problemas donde el espacio de búsqueda es no diferenciable o presenta discontinuidades. Además, la convergencia a la solución óptima depende de la elección de la solución inicial. Adicionalmente, los métodos clásicos de optimización son propensos a encontrar soluciones sub-óptimas en funciones multi-modales [15].

El cómputo evolutivo es un área de investigación en el campo de las ciencias de la computación que surgió como una alternativa a los métodos clásicos de optimización dado que resuelven algunas de las deficiencias que tienen estos últimos [15]. Esta área ha dado lugar a una gran variedad de algoritmos inspirados en el proceso de la evolución de las especies, generalmente conocidos como algoritmos evolutivos (AEs), los cuales son utilizados para aproximar soluciones de problemas de optimización.

Las principales diferencias entre los algoritmos evolutivos respecto a los métodos clásicos, son que los primeros utilizan una población de soluciones en lugar de utilizar una única solución, lo cual les permite escapar de los óptimos locales. Además, explotan únicamente la información proporcionada por las funciones objetivo y finalmente, en lugar de usar reglas de transición deterministas, los AEs emplean reglas de transición probabilísticas [22].

La idea principal de los algoritmos evolutivos es la siguiente: dado un entorno, existe una población de individuos los cuales tendrán que luchar por su supervivencia y reproduc-

ción. La aptitud de estos individuos será determinada por su entorno y por qué tan bien logren alcanzar sus objetivos. Esta idea básica se ajusta directamente a los problemas de optimización, ya que el problema a resolver puede ser visto como el entorno en el cual existe un conjunto de soluciones representadas como individuos cuya supervivencia dependerá de la calidad que presentan las mismas.

3.1. Conceptos preliminares

A continuación se presentan algunas definiciones necesarias antes de proporcionar la estructura general de un algoritmo evolutivo. Para obtener mayor información se recomienda al lector revisar [6, 18].

Definición 3.1.1. ***Individuo*** *Es un miembro de una población. Cada individuo representa una solución al problema a resolverse.*

Definición 3.1.2. ***Cromosoma*** *Es una estructura de datos que contiene una cadena de genes. Esta estructura de datos puede almacenarse, por ejemplo, como una cadena de bits.*

Definición 3.1.3. ***Esquema*** *Es un patrón de valores de genes de un cromosoma que puede incluir estados "no importa" denotados por* $*$*. Por ejemplo, usando un alfabeto binario, los esquemas usan el alfabeto* $0, 1, *$*, donde el último representa la presencia ya sea de* 0 *o de* 1*. Entonces, el esquema* $1*$ *puede instanciarse en los cromosomas* 11 *y* 10*.*

Definición 3.1.4. ***Orden de un esquema*** *Es el número de posiciones fijas de un esquema. Por ejemplo, el orden del esquema **1**0 es dos.*

Definición 3.1.5. ***Población*** *Es un conjunto de individuos que interactúan entre sí.*

Definición 3.1.6. ***Genotipo*** *Es la codificación de los parámetros que representan a una solución del problema a resolverse.*

Definición 3.1.7. ***Fenotipo*** *Es la decodificación del cromosoma, es decir, los valores obtenidos al pasar de la representación al valor obtenido al evaluar la función objetivo.*

Definición 3.1.8. ***Aptitud*** *Es el valor que se asigna a cada individuo, e indica qué tan bueno es un individuo respecto a los demás individuos en la población para la solución de un problema.*

Definición 3.1.9. ***Diversidad*** *Es una medida que indica de cierta manera que tan diferentes son los individuos de una población. Si los individuos no se parecen mucho entre sí (usando alguna métrica) se dice que la diversidad en la población es alta. De lo contrario, se dice que la diversidad en la población es baja. El rendimiento de un algoritmo evolutivo puede verse afectado, particularmente si la diversidad en la población es muy baja.*

Definición 3.1.10. ***Generación*** *Es el proceso en el cual se obtiene una nueva población de individuos a partir de una población existente mediante el uso de ciertos operadores evolutivos.*

Definición 3.1.11. ***Operador evolutivo*** *Es un mecanismo que afecta la forma en que se transmite información genética de una generación a la siguiente. Los operadores evolutivos principales son los siguientes:*

- *Cruza: Es un operador que es aplicado a dos o más individuos (en este contexto conocidos como padres) para producir uno o más individuos (conocidos como hijos).*
- *Mutación: Es un operador que aplica ligeras alteraciones al cromosoma de un individuo con el objetivo de producir un nuevo individuo. Un operador de mutación es siempre estocástico, es decir, la elección de qué partes del cromosoma serán afectadas, es un proceso aleatorio. Teóricamente, la mutación tiene el objetivo de conectar todo el espacio de búsqueda [18].*

Definición 3.1.12. ***Selección*** *Es el proceso en el que se decide qué individuos de una población serán seleccionados para su reproducción. Actúa como una fuerza que incrementa la calidad media de las soluciones en una población.*

Definición 3.1.13. ***Elitismo*** *Es un mecanismo utilizado en algunos algoritmos evolutivos para asegurar que los cromosomas de los miembros más aptos de una población pasen a la siguiente generación sin ser alterados por ningún operador genético. El uso de elitismo asegura*

que el individuo que posea la aptitud más alta no desaparezca en el proceso evolutivo. Sin embargo, el elitismo no necesariamente mejora la posibilidad de localizar el óptimo global [6].

Definición 3.1.14. ***Exploración*** *Es el proceso de visitar nuevas regiones del espacio de búsqueda, para ver si pueden encontrarse mejores soluciones. La exploración es buena para evitar quedar atrapado en óptimos locales.*

Definición 3.1.15. ***Explotación*** *Es el proceso de utilizar la información obtenida de los puntos visitados previamente para determinar qué lugares resulta más conveniente visitar. La explotación es útil para encontrar óptimos locales.*

El algoritmo 1 describe el proceso general de un algoritmo evolutivo. La idea general consiste en generar de manera aleatoria un conjunto de soluciones válidas las cuales serán evaluadas para determinar su capacidad para resolver un cierto problema. A continuación, comienza el proceso iterativo, el cual consiste en seleccionar de forma estocástica a los individuos que se utilizarán para producir una nueva generación. Ya seleccionados los individuos se aplicará el operador de cruza sobre ellos de tal forma que se obtenga un nuevo conjunto de individuos. Posteriormente, se aplicará mutación sobre la nueva generación y se evaluará su aptitud. Dependiendo del tipo de mecanismo de selección adoptado, se retendrá sólo a la nueva población (reemplazando totalmente a la anterior) o se realizará la unión de la nueva población con la anterior y se continúa con los mejores individuos (en términos de aptitud) de esta unión.
El proceso iterativo terminará hasta que se cumpla una cierta condición de paro. La condición más comúnmente utilizada es un número máximo de iteraciones o evaluaciones de la función objetivo.

Algoritmo 1 Algoritmo evolutivo

INICIALIZAR la población con soluciones aleatorias
EVALUAR a cada candidato
REPETIR HASTA (criterio de terminación se cumpla)
  SELECCIONAR a los padres a reproducirse
  APLICAR CRUZA sobre los padres seleccionados
  MUTAR a los nuevos individuos
  EVALUAR a los nuevos individuos
  SELECCIONAR a los individuos que formarán la próxima población
FIN REPETIR

3.1.1. Principales paradigmas

El estudio referente a los algoritmos evolutivos se ha dividido principalmente en tres paradigmas: los algoritmos genéticos, la programación evolutiva y las estrategias evolutivas. A continuación se describe de forma breve cada uno de estos paradigmas.

3.1.2. Algoritmo genético

El algoritmo genético es el paradigma evolutivo más conocido. Fue propuesto originalmente por John Holland [26] como un medio para estudiar el comportamiento adaptativo presente en la naturaleza. Sin embargo, actualmente es considerado como un método de optimización debido principalmente al título del libro publicado por Goldberg en 1989, llamado "Genetic Algorithms in Search, Optimization and Machine Learning" [22] y al éxito obtenido al resolver problemas de optimización [11].
La representantción de un individuo generalmente es mediante el uso de una cadena binaria. El operador con mayor importancia para la generación de nuevos individuos es la cruza (generalmente se utiliza la cruza de un punto), el mecanismo de selección de los padres es proporcional a la aptitud de los individuos. La mutación actúa como un operador secundario y debido a que el individuo es representado mediante una cadena binaria, este operador es simplemente el complemento de ciertas posiciones seleccionadas de forma

aleatoria. Finalmente, todos los padres serán reemplazados por los hijos en la próxima generación. Esto se puede observar en el pseudocódigo de un algoritmo genético mostrado en el algoritmo 2.

Algoritmo 2 Algoritmo genético

INICIALIZAR una población de individuos representados mediante cadenas binarias aleatorias
EVALUAR a cada individuo
REPETIR HASTA (criterio de terminación se cumpla)
  SELECCIONAR a los padres a reproducirse con base en su aptitud
  APLICAR CRUZA sobre los padres seleccionados con cierta probabilidad pc
  MUTAR a los nuevos individuos con cierta probabilidad pm
  EVALUAR a los nuevos individuos
  REEMPLAZAR a la generación actual con la nueva generación
FIN REPETIR

3.1.3. Programación evolutiva

La programación evolutiva fue propuesta por Lawrence Fogel en los sesentas y utiliza el concepto de la evolución de las especies como un proceso de aprendizaje con el objetivo de generar una inteligencia artificial [10, 20, 19]. La inteligencia en este contexto es vista como la capacidad que tiene un sistema de adaptar su comportamiento de tal forma que logre cumplir con sus objetivos. En la actualidad, la programación evolutiva es utilizada como un método de optimización inspirado en el proceso de evolución a nivel de especies a diferencia del algoritmo genético que está basado en el principio de evolución a nivel del individuo. Por tal motivo, cada individuo en la población es visto como una especie, por lo que el concepto de cruza es omitido ya que dos especies diferentes no se pueden cruzar con el objetivo de reproducirse. Bajo este enfoque, una nueva generación de especies es producida mediante la mutación. El mecanismo de supervivencia consiste en un torneo entre la unión de la población de padres con la población de hijos (los individuos de mayor aptitud tendrán una mayor posibilidad de ganar este torneo). Esto se puede observar en el pseudocódigo mostrado en el algoritmo 3.

Algoritmo 3 Programación evolutiva

INICIALIZAR una población de individuos representados mediante un vector de valores reales.
EVALUAR a cada individuo
REPETIR HASTA (criterio de terminación se cumpla)
  APLICAR MUTACIÓN con el objetivo de formar una nueva población de individuos
  EVALUAR a los nuevos individuos
  REEMPLAZAR a la generación utilizando un torneo estocástico de la unión entre la generación actual y la nueva generación
FIN REPETIR

3.1.4. Estrategias evolutivas

Las estrategias evolutivas, fueron propuestas por tres estudiantes de la Universidad Técnica de Berlín (Peter Bienert, Ingo Rechenberg y Hans-Paul Schwefel) en los sesentas, con el objetivo de optimizar la forma de un tubo curvo [46].

En este caso, un individuo es representado mediante un vector de números reales. La mutación es normalmente realizada mediante la suma de un número aleatorio con una distribución normal a cada componente del vector que conforma al individuo. La distribución normal tiene una media de cero y una desviación estándar σ, donde σ es el tamaño de paso. En la literatura de las estrategias evolutivas, existe una gran cantidad de esquemas para seleccionar a los individuos que conformarán la siguiente generación; el más sencillo de ellos es el esquema denotado $(1+1)-ES$, en el cual un nuevo individuo es generado por medio de la mutación y pasa a sustituir al padre si la aptitud de éste es mejor. Otra alternativa es el esquema denotado como $(1,1)-ES$, en el cual siempre reemplazan a los padres con los hijos generados por ellos.

Una adición importante a las estrategias evolutivas es la regla de $1/5$ propuesta por Rechenberg [46], la cual ajusta el tamaño de paso en la mutación. La regla indica que si más de uno de cada cinco pasos fueron exitosos al aplicar la mutación (es decir, que la mutación produjo hijos más aptos que sus padres), entonces la mutación está en riesgo de tomar pasos muy pequeños (explotación), y σ necesita ser incrementado. De lo contrario, si cae por debajo de $1/5$, entonces la mutación está tomando pasos muy grandes (exploración) por lo que el valor de σ necesita ser disminuido [28]. Por ende, la tasa de éxito ideal

es precisamente $1/5$.

3.2. Algoritmos evolutivos multi-objetivo

Como se discutió al inicio de este capítulo, una de las principales diferencias entre las técnicas clásicas de optimización respecto a los algoritmos evolutivos, es que los segundos usan una población de soluciones durante el proceso de optimización. Esta diferencia brinda una gran ventaja a los algoritmos evolutivos al momento de resolver problemas de optimización multi-objetivo. Dado que el principal interés en la optimización multi-objetivo es producir un conjunto de soluciones compromiso, debería ser posible realizar ciertos cambios a un algoritmo evolutivo de tal forma que sea posible obtener una población de soluciones no dominadas durante una ejecución.

Por lo tanto, es posible establecer un esquema general para un algoritmo evolutivo multi-objetivo utilizando un proceso similar al descrito en la sección anterior, pero ahora utilizando el concepto de optimalidad de Pareto como mecanismo de comparación entre los individuos de la población y a la vez incorporando algún mecanismo que asegure la diversidad y distribución de las soluciones a lo largo del frente de Pareto. El algoritmo 4 descrito en [34] muestra un esquema general para un algoritmo evolutivo multi-objetivo (AEMO).

3.2.1. Elitismo

Durante el proceso evolutivo de un AEMO, se pueden distinguir dos poblaciones de individuos que aproximan el frente de Pareto, el $P_{conocido}$ que generalmente se denomina población secundaria o archivo externo y el P_{actual} que se denomina población primaria. $P_{conocido}(t)$ es un conjunto que se actualiza en cada generación, de forma que mantiene las mejores soluciones (no dominadas) encontradas hasta la generación t y $P_{actual}(t)$ es el conjunto de soluciones no dominadas obtenidas en la generación t.

Usualmente, el conjunto $P_{conocido}$ es actualizado al final de cada generación de la siguiente forma: $P_{conocido}(t) = P_{conocido}(t-1) \cup P_{actual}(t)$. Posteriormente, se deberán filtrar las soluciones no dominadas de este nuevo conjunto obtenido, dado que las soluciones son

Algoritmo 4 Algoritmo evolutivo multi-objetivo genérico

Entrada: Parámetros particulares del AEMO
Salida: Un conjunto S que aproxima el frente de Pareto para un POM
Generar de forma aleatoria una población de individuos $P_t, t = 0$
Para cada individuo $p \in P_t$
Evaluar las funciones objetivo
Asignar el valor de aptitud o jerarquía correspondiente
Fin Para
Mientras el críterio de terminación no se alcance
Seleccionar $P'_t \subset P_t$ basándose en la aptitud
Producir una nueva generación P''_t por medio de los operadores de cruza y mutación
Para cada individuo $p'' \in P''_t$
Evaluar las funciones objetivo
Asignar el valor de aptitud o jerarquía correspondiente
Fin Para
Seleccionar los individuos que pasarán a la siguiente generación de $P_t \cup P''_t$ de forma que se conforme P_{t+1}
t = t + 1
Fin Mientras

no dominadas respecto al conjunto al que pertenecen.

El tamaño del archivo puede crecer de forma indefinida debido a que en cada generación son agregadas las nuevas soluciones obtenidas. Por tal motivo, se establece un tamaño máximo que éste puede llegar a alcanzar.

Como es mencionado en [7], cualquier implementación de un AEMO debería incluir una población secundaria compuesta por todas las soluciones no dominadas encontradas hasta cierto momento ($P_{conocido}(t)$). Esto es debido principalmente a la naturaleza estocástica de los AEMOs que no garantiza que las mejores soluciones encontradas permanezcan en la población al finalizar el proceso evolutivo.

3.3. Optimización mediante cúmulos de partículas

La optimización mediante cúmulos de partículas (PSO por sus siglas en ingles), fue propuesta en 1995 por James Kennedy y Rusell Eberhart [30]. Es una metaheurística inspirada en el comportamiento social presente en diversas especies de animales.

La idea general de un PSO consiste en una población de partículas que se mueven sobre

todo el espacio de búsqueda obedeciendo unas reglas simples que alteran tanto su posición como su velocidad.

3.3.1. PSO visto en términos sociales y cognitivos

En esta sección se describe de forma breve a un PSO desde el punto social y cognitivo. En [32] se presenta un análisis detallado del comportamiento social presente en los individuos y cómo es que la interacción de éstos puede resolver distintas clases de problemas.
Se asume la existencia de una población de individuos simples denominados agentes, los cuales interactúan entre ellos con el objetivo de resolver un problema. Cada agente puede ser descrito mediante tres principios fundamentales:

- **Evaluación**: es la tendencia que tiene un agente a evaluar cualquier estímulo en su entorno como positivo o negativo. El aprendizaje no puede ocurrir a menos que el agente sea capaz de evaluar las características de su entorno. Desde este punto de vista, el aprendizaje puede ser definido como cualquier cambio que permita al agente mejorar la evaluación promedio de su entorno.

- **Comparación**: un agente tiende a comparar algunas de las características o acciones que lo identifican respecto a las de otros agentes desde un punto de vista crítico y tiende a imitar sólo a aquellos que le permitan mejorar la evaluación promedio de su entorno.

- **Imitación**: la imitación es una forma de aprendizaje. Sin embargo, muy pocas especies de animales son capaces de imitar de forma auténtica. Un humano, no sólo es capaz de imitar la acción de otro humano, también es capaz de comprender el propósito por el cual fue realizada tal acción. Un agente es capaz de imitar a otro agente sin necesidad de comprender el motivo por el cual se realizó la acción.

De esta forma, un PSO puede verse como un conjunto de agentes, donde cada agente se encuentra en movimiento y tiene que decidir cuál será la próxima dirección a seguir. La única información disponible es su propia experiencia hasta ese momento, qué tan buena

resultó dicha posición (evaluación) y cómo les fue a los demás agentes que están a su alrededor (comparación).
Finalmente, la dirección en la cual se moverá cada agente consistirá en la combinación de tres direcciones: la dirección actual en la que mueve dicho individuo, la dirección que representa a su mejor experiencia y la dirección que representa al éxito obtenido por los demás agentes a su alrededor (imitación).

3.3.2. Descripción de la metaheurística

A continuación se presentan algunas definiciones necesarias antes de presentar la estructura general de un PSO.

Definición 3.3.1. ***Cúmulo*** *Es el equivalente a la población de individuos en un algoritmo evolutivo.*

Definición 3.3.2. ***Partícula*** *Un individuo del cúmulo. Cada partícula representa una solución potencial al problema a ser resuelto. Una partícula está compuesta por una posición en el espacio de búsqeda, una dirección que indica la trayectoria que esta partícula sigue y un registro histórico con la mejor posición en la que la partícula ha estado.*

Definición 3.3.3. ***Velocidad*** *Es el vector que indica la dirección en la que una partícula se deberá mover.*

Definición 3.3.4. ***Factores de aprendizaje*** *Representan la tendencia que tiene una partícula a seguir el éxito de la mejor partícula en el cúmulo o a seguir su propio éxito, representados por c_1 (parámetro social) y c_2 (parámetro cognitivo). Generalmente los valores de dichos factores son establecidos a un inicio y la suma de ambos no debe ser mayor a cuatro [29].*

Definición 3.3.5. ***Mejor posición local*** *Es el registro histórico de la experiencia de una partícula y representa la posición con mejor aptitud en la que ha estado dicha partícula, usualmente denotado como $lbest$.*

Definición 3.3.6. ***Mejor posición global*** *Es la posición de la partícula con mejor aptitud de todo el cúmulo, usualmente denotado como $gbest$.*

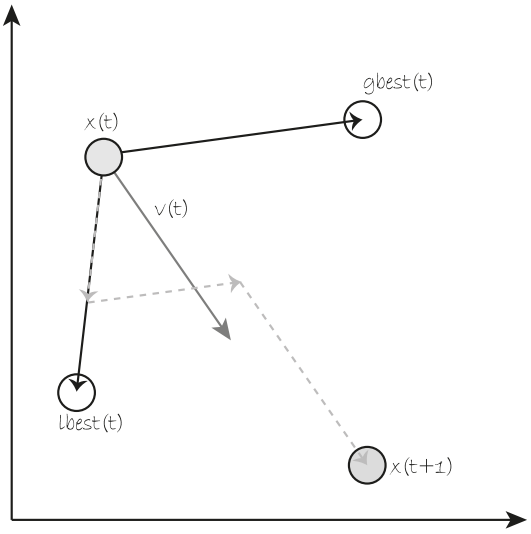


Figura 3-1: El movimiento de una partícula se puede ver como la suma de tres vectores.

Cada partícula afecta su trayectoria de vuelo tomando en consideración su propia experiencia de vuelo, y la experiencia de vuelo de sus vecinos. Supongamos que $x_i(t)$ denota la posición de la partícula i, en un instante de tiempo t y $v_i(t)$ denota la velocidad de la partícula i en el instante de tiempo t. Entonces, la posición de la partícula i en el instante de tiempo $t+1$, está definida mediante:

$$
\begin{aligned}
v_i(t+1) &:= v_i(t) + c_1 * rand_1 \times (gbest - x_i(t)) + c_2 * rand_2 \times (lbest_i(t) - x_i(t)) \\
x_i(t+1) &:= x_i(t) + v_i(t+1)
\end{aligned}
\tag{3.1}
$$

Donde $rand_1, rand_2 \in [0,1]$. En otras palabras, el movimiento de una partícula i con posición $x_i(t)$ y velocidad $v_i(t)$ puede verse como la suma de 3 vectores: el primer vector (v_i) conserva la trayectoria actual que lleva la partícula i, el segundo vector ($c_1 * rand_1 \times (gbest - x_i(t))$) representa un vector con una magnitud aleatoria que va desde $x_i(t)$ hasta $gbest$ y el tercer vector ($c_2 * rand_2 \times (lbest_i(t) - x_i(t))$ representa un vector con una magnitud aleatoria que va desde $x_i(t)$ hasta $lbest$. Esto se puede ver de forma gráfica en la figura 3-1. Por lo tanto, un PSO intenta balancear la exploración y explotación combinando métodos de búsqueda local y global. En este aspecto, un PSO es similar a un algoritmo genético y a los algoritmos meméticos [40]. Finalmente, es posible presentar la estructura

básica de un PSO, como se puede observar en el algoritmo 5.

Algoritmo 5 Pseudo-código del PSO

Entrada: n : número de partículas
  Para cada partícula $x_i,\ i \in [1, ..., n]$
    Inicializar x_i de forma aleatoria
    Inicializar el vector de velocidad asociado $v_i := 0$
    Establecer inicialmente $lbest_i := x_i$
    Si $f(lbest_i) < f(gbest)$ **entonces**
      $gbest := lbest_i$ { Obtener la mejor posición global}
    Fin Si
  Fin Para
  Mientras el criterio de terminacipn no se alcance
    Para cada partícula $x_i,\ i \in [1, ..., n]$
      Actualizar la velocidad de la partícula i respecto a la siguiente fórmula
      $v_i := v_i + c_1 * rand() \times (gbest - x_i) + c_2 * rand() \times (lbest_i - x_i)$
      Actualizar la posición de la partícula i respecto a la siguiente fórmula
      $x_i := x_i + v_i$
      Si $f(x_i) < f(lbest_i)$ **entonces**
        $lbest_i := x_i$ { Actualizar la mejor posición de la partícula}
        Si $f(lbest_i) < f(gbest)$ **entonces**
          $gbest := lbest_i$ { Actualizar la mejor posición del cúmulo}
        Fin Si
      Fin Si
    Fin Para
  Fin Mientras

3.3.3. PSO visto como un algoritmo evolutivo

Como argumentan los autores del PSO en [17], un PSO puede verse como un algoritmo evolutivo, en el que los individuos de una población son abstraídos al concepto de partículas. Cada partícula representa una solución candidata al problema a ser resuelto. Aunque no existe un operador de cruza, el concepto está presente en un PSO dado que cada partícula oscila sobre el promedio ponderado entre el mejor local ($lbest$) y el mejor global ($gbest$). La mutación es el único operador presente, y es el responsable del cambio en la dirección de cada partícula. Finalmente, los individuos de un PSO no tienen que competir entre sí para asegurar su supervivencia. Ésta es la principal diferencia entre los demás algoritmos evolutivos y un PSO, ya que desde un inicio, el PSO fue diseñado para simular

el proceso colaborativo presente entre los individuos de la población.

Uso de inercia y constricción

Cuando un PSO es ejecutado sin limitar la velocidad de sus partículas, éstas tienden a oscilar fuera de la región de interés. Existen tres alternativas principales para solucionar este problema:

1. Limitar la velocidad utilizando una constante *Vmax*:
   Para cada dimensión $j \in [0, d]$
   si $v_{i,j} > Vmax$ **entonces** $v_{i,j} = Vmax$
   de lo contrario, si $v_{i,j} < -Vmax$ **entonces** $v_{i,j} = -Vmax$

2. Peso de inercia (w): propuesto por Shi and Eberhart en mayo de 1998 [48]. Proporciona un equilibrio entre la búsqueda local y la global, incrementando la posibilidad de encontrar el óptimo global en un número razonable de iteraciones.

   $$v_i := w \times v_i + c_1 * rand() \times (gbest - x_i) + c_2 * rand() \times (lbest_i - x_i)$$

   Cuando w toma valores pequeños, $w < 0.8$, el PSO funciona como un buscador local; si exite una solución aceptable dentro del espacio de búsqueda, el PSO será capaz de encontrar el óptimo global; de lo contrario, no podrá hacerlo. Si $w > 1.2$, el PSO funciona como un buscador global e incluso intentará explotar las nuevas áreas descubiertas. Los autores recomiendan empezar la búsqueda con un valor de 1.4 y decrementar su valor de forma gradual hasta llegar a 0.5.

3. Factor de constricción: Clerc and Kennedy analizaron la trayectoria de una partícula en un intervalo de tiempo tanto discreto como continuo y propusieron el uso de coeficientes de constricción, con el objetivo de controlar la explosión que ocurre en la velocidad [4].

   $$\chi := \frac{2\kappa}{2 - (c_1 + c_2) - \sqrt{(c_1 + c_2)^2 - 4(c_1 + c_2)}}$$

$$v_i := \chi \left(v_i + c_1 * rand() \times (gbest - x_i) + c_2 * rand() \times (lbest_i - x_i) \right)$$

La variable $\kappa \in [0, 1]$ controla la amplitud de la constricción; usualmente $\kappa = 1$. Cabe destacar que, utilizando el factor de constricción, la suma de c_1 con c_2 debe ser mayor a 4; usualmente $c_1 = 2.05$ y $c_2 = 2.05$ [32].

Con el objetivo de determinar cuál es la mejor alternativa, Eberhart y Shi [49] realizaron una comparativa entre los pesos de inercia y los factores de constricción utilizando cinco funciones de prueba y concluyeron que el mejor enfoque es utilizar el factor de constricción a la vez que se limita la velocidad Vmax al rango máximo de la variable.

3.3.4. Topologías

Como ya se ha comentado, las partículas tienden a ser influenciadas por el éxito de las partículas con las que están comunicadas. No es necesario que las conexiones se establezcan entre partículas cercanas, más bien es posible conectarlas con otras partículas que se encuentren en su vecindario. Una topología establece la forma en que se comunican las distintas partículas en una población. Entre las principales topologías se encuentran las siguientes:

Topología lbest

La topología lbest fue propuesta como una manera de lidiar con problemas difíciles. En esta topología, cada partícula está conectada con sus k vecinos inmediatos en el arreglo de la población. Si $k = 2$, entonces se tiene una topología similar a un anillo en donde cada partícula sólo puede interactuar con sus vecinos a su izquierda y derecha. La principal ventaja de esta topología es que puede crear una especie de sub-poblaciones que se enfocan en buscar en distintas regiones del espacio de búsqueda.

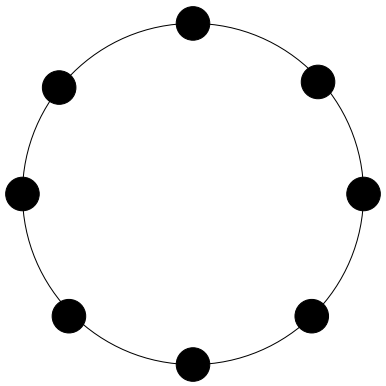


Figura 3-2: Ejemplo de una topología $lbest$ con $k = 2$ utilizando ocho partículas

Topología completa

También conocida como topología gbest. En esta topología todas las partículas están conectadas entre sí, por lo que todas las partículas dirigirán su trayectoria hacia la mejor partícula del cúmulo. Esta topología tiende a converger más rápido que la mayoría de las topologías, sin embargo, las partículas pueden quedar atrapadas fácilmente en un óptimo local.

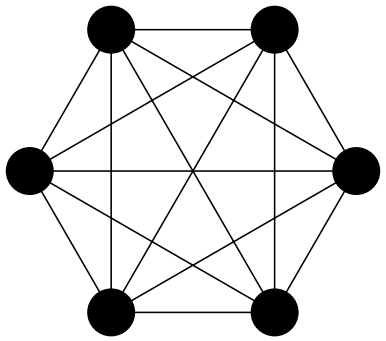


Figura 3-3: Ejemplo de una topología completa utilizando seis partículas.

Topología von Neumann

La topología von Neumann, comprende una especie de matriz de tamaño $n \times m$, en la cual cada partícula está conectada con sus vecinos situados en el norte, sur, este y oeste. Kennedy y Mendes [31] analizaron los efectos de distintas topologías y descubrieron, por ejemplo, que la topología de von Neumann presenta mejores resultados que las topologías estándar en un conjunto de problemas de prueba usados para optimización global.

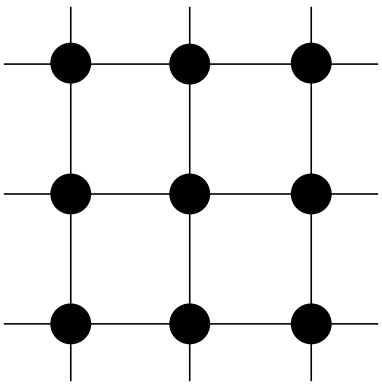


Figura 3-4: Ejemplo de una topología von Neumann utilizando nueve partículas.

Entonces se puede establecer que la diversidad de una población se puede incrementar reduciendo el número de informantes que tiene cada partícula, mientras que la propagación de los resultados se vuelve más rápida y completa conforme el número de informantes se incrementa.

Existe evidencia que sugiere que los tamaños de población muy grandes son buenos en problemas de alta dimensionalidad y que las topologías altamente conectadas funcionan mejor en problemas unimodales, mientras que las topologías ligeramente conectadas son superiores en problemas muti-modales [43].

3.3.5. Optimización mediante cúmulos de partículas para problemas de optimización multiobjetivo

Al extender un PSO para resolver problemas de optimización multi-objetivo, se espera que éste sea capaz de obtener un conjunto de soluciones no dominadas en cada ejecución. Sin embargo, dada su naturaleza estocástica, es necesario utilizar un archivo externo que almacene las soluciones no dominadas encontradas durante el proceso de búsqueda. De forma similar a otros AEMOs, existen tres cuestiones principales a tomarse en cuenta al extender un PSO para resolver problemas multi-objetivo [51].

1. ¿Cómo seleccionar las partículas que servirán como líderes, de forma que atraigan a las demás partículas hacia el frente de Pareto?

2. ¿Cómo retener todas las soluciones no dominadas encontradas durante el proceso de búsqueda?

3. ¿Cómo mantener la diversidad en el cúmulo de tal forma que se evite la convergencia hacia una única solución?

El algoritmo 6 presenta la estructura general de un optimizador multi-objetivo mediante cúmulos de partículas (MOPSO, por sus siglas inglés). Como se puede ver, se utiliza un archivo externo para retener todas las soluciones no dominadas y un operador de mutación como mecanismo para mantener la diversidad en el cúmulo. Dado que el archivo externo es un conjunto de soluciones no dominadas, no es posible establecer una relación de orden sobre éstas, por lo que generalmente se asume que cualquier solución puede servir como líder.

Algoritmo 6 Pseudo-código general de un MOPSO

1 Inicializar cúmulo
2 Iniciar el archivo de líderes
3 Determinar la calidad de los líderes
REPETIR HASTA (criterio de terminación se cumpla)
  Para cada partícula en el cúmulo
    4 Seleccionar el líder
    5 Actualizar la posición de la partícula
    6 Aplicar mutación (turbulencia)
    7 Evaluar la partícula
    8 Actualizar el mejor personal de la partícula utilizando dominancia de Pareto
  Fin Para
  9 Actualizar el archivo de líderes
  10 Determinar la calidad de los líderes
FIN REPETIR

3.4. Estado del arte en AEMOs y MOPSOs

En esta sección se describen brevemente un AEMO y dos MOPSOs del estado del arte que se usarán para comparar resultados respecto al algoritmo compacto multi-objetivo propuesto en esta tesis.

MOEA/D

El *Multi-Objective Evolutionary Algorithm based on Decomposition* fue propuesto por Zhang y Li [55], y utiliza el concepto de descomposición, el cual consiste en transformar un problema de optimización multi-objetivo en un conjunto de problemas mono-objetivo mediante el uso de un conjunto de vectores de ponderación distribuidos de forma uniforme sobre el espacio objetivo, los cuales optimiza de forma simulťanea. Cada subproblema es optimizado utilizando únicamente la información de subproblemas vecinos. El vecindario de un vector de ponderación λ_i es el conjunto formado por los T vectores de ponderación más cercanos[1] a éste. El proceso evolutivo consiste en generar una nueva solución por medio de los operadores genéticos utilizando dos individuos al azar pertenecientes al vecindario de λ_i $\forall i \in [1, n]$. Posteriormente, se compara dicha solución respecto a las soluciones en su vecindario para determinar si las puede reemplazar (esto sucede porque en teoría, la solución no es epecífica a un problema de métricas ponderadas. Más bien esta solución puede funcionar para cualquier problema de métricas ponderadas dentro del vecindario). Finalmente, se eliminan todas las soluciones del archivo que sean dominadas por esta nueva solución y se agrega al archivo si dicha solución no es dominada por ninguna solución dentro del archivo.

dMOPSO

El optimizador mediante cúmulos de partículas multi-objetivo *A Multi-objective Particle Swarm Optimizer Based on Decomposition* fue propuesto por Zapotecas y Coello [54] y utiliza un enfoque similar al empleado por MOEA/D, ya que adopta un conjunto de vectores de ponderación para descomponer un problema multi-objetivo en múltiples sub-

[1] Más cercanos mediante el uso de una distancia euclidiana.

Algoritmo 7 Pseudo-código de MOEA/D

Entrada: F: función a optimizar, m: número de funciones objetivo, N: tamaño de la población, Un conjunto de vectores de ponderación $\{\lambda_1, ..., \lambda_N\}$, T: tamaño del vecinadario

Salida: EP: archivo que contendrá una aproximación al frente de Pareto

1 $EP = \emptyset$

2 Construir la matriz B **de tamaño** $N \times T$**:** Donde $B(i)$ está compuesto por los índices de los vectores de ponderación más cercanos a λ_i, $B(i) = \{i_1, ..., i_T\}$, donde $\lambda^{i_1}, ..., \lambda^{i_T}$

3 Generar una población de soluciones de forma aleatoria: $x^1, ..., x^N$

4 Evaluar a la población de soluciones

5 Inicializar el vector de referencia: $z_i = \infty,\ \forall i \in [1, m]$

REPETIR HASTA (criterio de terminación se cumpla)

Para $i = 1$; hasta n

6 Reproducción Seleccionar dos índices k, l de forma aleatoria de $B(i)$, y generar una nueva solución y por medio de operadores genéticos.

7 Reparar Aplicar un mecanismo específico del problema para reparar la solución y producir y'

8 Actualizar el vector de referencia: si $f_j(y') < z_j : z_j = f_j(y'),\ \forall j \in [1, m]$

9 Actualizar las soluciones vecinas: $\forall j \in B(i)$, **si** $g^{te}(y'|\lambda^j, z) \leq g^{te}(x^j|\lambda^j, z)$ entonces $x^j = y'$

10 Eliminar todas las soluciones en EP que sean dominadas por y'**, y agrega la solución** y' **al archivo EP** si y solo sí esta solución no es dominada

Fin Para

FIN REPETIR

problemas mono-objetivo usando una función de escalarización de Tchebycheff. A diferencia de MOEA/D, no se utiliza el concepto de optimalidad de Pareto para aproximar el conjunto de soluciones hacia el frente de Pareto y tampoco requiere de un archivo externo para almacenar las soluciones. Este algoritmo utiliza un archivo de líderes que servirá como guía durante el proceso de búsqueda. Utiliza el mecanismo empleado por MOPSO/D [42] para actualizar al mejor personal de una partícula. En lugar de utilizar el concepto de mutación (perturbación) asigna una edad a cada partícula. Si después de un determinado número de iteraciones una partícula no es capaz de mejorar su mejor personal, entonces es reemplazada utilizando un mecanismo similar al adoptado por BB-PSO [29]. El algoritmo 8 muestra el pseudo-código de dMOPSO.

SMSPSO

Después de analizar el desempeño de cinco MOPSOs en [16], Nebro et al, descubrieron que eran incapaces de resolver problemas multi-frontales de forma satisfactoria. Al analizar la razón de este problema descubrieron que la velocidad de las partículas se tornaba muy alta, lo cual causaba que las partículas oscilaran fuera de la región de interés. Tomando como punto de partida al algoritmo con mejores resultados (OMOPSO [50]) desarrollaron un nuevo algoritmo al que llamaron SMPSO [38] (Speed-constrained Multi-objective PSO).
Este algoritmo incorpora el factor de constricción como mecanismo para limitar la velocidad de las partículas y limitan la velocidad de cada partícula al valor medio entre el límite superior y el inferior de las variables. Finalmente, reemplazan los operadores de mutación originalmente propuestos en OMOPSO por la mutación polinomial [15]. El algoritmo 9 muestra el pseudo-código de esta propuesta. La selección de líderes consiste en un torneo binario entre las soluciones del archivo, tomando en cuenta la distancia de agrupamiento propuesta originalmente para NSGA II [12]. Finalmente, cabe destacar el uso simultáneo del factor de contricción con el peso inercia para controlar la velocidad de una partícula.

Algoritmo 8 Pseudo-código de dMOPSO

1 Generar un conjunto de vectores de ponderación bien distribuidos $W = \{w_1, ..., w_N\}$
2 Generar una población de soluciones de forma aleatoria: $\mathcal{P}^t = x_1, ..., x_N$
3 Asignar tanto la velocidad como la edad de cada partícula en cero: $v_i^t = a_i^t = 0 \; \forall i \in [1, N]$
4 Definir el mejor personal $lbest_i = x_i \; \forall i \in [1, N]$
5 Definir al conjunto Gbest como la población inicial $Gbest(t) = \mathcal{P}^t$
REPETIR HASTA (criterio de terminación se cumpla)
  6 Revolver al conjunto Gbest
  Para $i = 1$; hasta n
    Si $a_i < T_a$ **entonces**
      7 Actualizar velocidad utilizando como gbest al iésimo elemento en $Gbest(t)$
    Si no
      7 Reiniciar partícula, establecer $a_i = v_i = 0$ y generar x_i utilizando un valor aleatorio con distribución normal con media $\frac{gbest_i - lbest_i}{2}$ y desviación estándar $|gbest_i - lbest_i|$, para cada dimensión de la partícula.
    Fin Si
    8 Reparar los límites Ajustar los límites tanto de las variables como de la velocidad.
    9 Actualizar el vector de referencia: si $f_j(x_i) < z_j : z_j = f_j(x), \; \forall j \in [1, k]$
    10 Actualizar el mejor personal
    Si $g(x_i|\lambda^j, z) \leq g(lbest_i|\lambda^j, z)$ **entonces**
      $lbest_i = x_i, a_i = 0$
    Si no
      $a_i = a_i + 1$
    Fin Si
    11 Actualizar Gbest de la unión $Gbest \cup \mathcal{P}^{t+1}$. Retener las N mejores soluciones que optimicen los sub-problemas asociados a los vectores de ponderacíon.
  Fin Para
FIN REPETIR

Algoritmo 9 Pseudo-código de SMPSO

1 Inicializar el cúmulo
2 Inicializar el archivo de líderes
3 generación = 0
Mientras generación < máxGeneraciones
4 Actualizar la velocidad de cada partícula $i \in [0, \texttt{tamCúmulo}]$ **respecto a la siguiente fórmula**
$v_i := \chi(w \times v_i + c_1 * rand() \times (gbest - x_i) + c_2 * rand() \times (lbest_i - x_i))$
5 Actualizar la posición de la partícula utilizando la siguiente fórmula
$x_i := x_i + v_i$
6 Aplicar mutación polinomial
7 Evaluar al cúmulo
8 Actualizar el archivo de líderes
9 Actualizar la memoria de las partículas
Fin Mientras
10 Devolver el archivo de líderes

Capítulo

4

Metaheurísticas compactas

En este capítulo se discuten las principales metaheurísticas compactas. La idea general de una metaheurística compacta es reemplazar la población de un algoritmo evolutivo por un vector de probabilidades. Los individuos del algoritmo son generados por medio del vector de probabilidades y servirán como guía para actualizar los valores de dicho vector.

4.1. Algoritmo genético compacto

Determinar el tamaño adecuado de la población para un algoritimo genético, ha sido un problema ampliamente estudiado desde los orígenes de la investigación en el área. Si el tamaño de la población es muy pequeño, es improbable que el algoritmo genético pueda encontrar una buena solución al problema [23]. Sin embargo, si el tamaño de población es muy grande, el algoritmo genético desperdiciará gran parte del tiempo procesando individuos innecesarios [24].

Con el fin de derivar una ecuación para el tamaño óptimo de población, Harik et al. [24] propusieron un modelo que relaciona a un algoritmo genético con una caminata aleatoria en una dimensión. Para esto, identificaron dos factores que influyen de manera significativa en la calidad de las soluciones de un algoritmo genético. Estos factores son la fuente inicial de bloques constructores[1]y la selección de los mejores bloques constructores sobre

[1]Para estos fines, un bloque constructor es definido como: el esquema de orden mínimo que contribuye

sus competidores[2].

Dado que no existen interacciones sobre los bloques constructores, los autores consideraron que es posible resolver cada bloque constructor de forma independiente, por lo que su modelo se enfocó en resolver un bloque constructor a la vez. La idea general de su modelo es la siguiente. En la población inicial de un algoritmo genético, existen algunas instancias de un bloque constructor[3]. Durante la ejecución del algoritmo, el número de instancias puede aumentar o disminuir. Eventualmente, el bloque constructor se extenderá a través de todos los miembros de la población, o se extinguirá.

Esta dinámica sugirió a los autores la posibilidad de simular de forma directa el comportamiento de su modelo para problemas de orden-uno[4], dando origen al algoritmo genético compacto [25].

El algoritmo genético compacto representa a la población mediante un vector de probabilidades cuyos valores iniciales son establecidos en 0.5. En cada iteración se generan dos individuos que compiten para establecer a un ganador. Posteriormente, se modificarán ligeramente los valores del vector de probabilidades de forma que asimilen los valores de los genes del individuo ganador. El proceso completo es mostrado en el algoritmo 10.

4.1.1. Selección y cruza

La selección tiene como objetivo propagar los mejores bloques constructores sobre los individuos de la población. Sin embargo, no siempre es posible propagar los mejores genes. Esto se debe a que los genes son evaluados dentro del contexto del individuo y es posible que la contribución de otros genes tenga un mayor valor provocando una decisión incorrecta. Como ejemplo de este caso, se intenta decidir sobre dos individuos A y B, cúal tiene mejor aptitud para el problema onemax, que consiste en encontrar al cromosoma con mayor cantidad de unos. La tabla 4.1 muestra el cromosoma y la aptitud de ambos in-

a encontrar al mínimo global

[2]Se espera que el mecanismo de selección sea capaz de elegir a los individuos con los bloques constructores correctos omitiendo a los demás individuos.

[3]Los autores consideraron que la única fuente de bloques constructores en un algoritmo genético es la inicialización aleatoria de la población.

[4]Un problema de orden-uno, es referido por los autores como aquél que puede ser resuelto mediante la combinación de esquemas de orden-uno.

Individuo	Cromosoma	Aptitud
A	110	2
B	001	1

Tabla 4.1: Aptitud tanto del individuo A como del individuo B para el problema onemax

dividuos. Al realizar una competencia entre A y B, es claro que el individuo A tiene mejor aptitud. Sin embargo, se realizó un error de selección a nivel del gene, donde el tercer gene del individuo B claramente contribuye de mejor manera que el del individuo A. En otras palabras, la selección prefiere al esquema **0 que al esquema **1. El papel de la población es el de amortiguar un número finito de malas decisiones.

En el algoritmo genético compacto, el valor $1/n$ es responsable de simular el efecto que tiene un algoritmo genético sobre una población de tamaño n.

El rol de la cruza sobre un algoritmo genético, es el combinar las mejores características de un individuo. Sin embargo, el uso repetido de este operador puede causar una decorrelación entre los genes de una población. Por lo tanto, la generación de nuevos individuos por medio del vector de probabilidades puede ser vista como un atajo para el objetivo eventual de la cruza [25]. Los autores argumentan que la generación de nuevos individuos en el algoritmo genético compacto es equivalente a realizar un número infinito de cruzas por generación en un algoritmo genético tradicional, por lo que ningún gen de un algortitmo genético compacto tiene relación con otro, a diferencia del algoritmo genético tradicional donde los genes aún conservan una pequeña cantidad de correlación.

4.1.2. Simulación de una presión de selección mayor

El poder simular mayores tasas de selección, debería permitir al algoritmo genético compacto resolver problemas que requieran bloques constructores de ordenes superiores, de una forma similar a la que realiza un algoritmo genético tradicional con cruza uniforme. El añadir una mayor presión de selección, puede compensar los efectos disruptivos de la cruza uniforme.

Es posible aumentar la presión de selección mediante una modificación al algoritmo genético compacto que simule el efecto de un torneo de tamaño s. El siguiente mecanismo

produce dicho efecto: 1) Generar s individuos por medio del vector de probabilidades y determinar al ganador. 2) Realizar una competencia entre el ganador y los $s-1$ individuos restantes. Claramente, el ganador influirá $s-1$ veces sobre el vector de probabilidad.

4.2. Algoritmo genético compacto con elitismo persistente y no persistente

Al resolver problemas sencillos (por ejemplo problemas unimodales continuos) involucrando bloques constructores de orden inferior, el algoritmo genético compacto tiene un desempeño similar al del algoritmo genético tradicional con cruza uniforme. Sin embargo, el algoritmo compacto no es capaz de producir soluciones aceptables cuando se enfrenta a problemas difíciles (por ejemplo problemas deceptivos o problemas multimodales) dado que no tiene una memoria para retener el conocimiento requerido (por ejemplo, errores de decisión o información de vinculación entre genes) sobre la no linealidad de los problemas [1].

Para obtener mejores resultados en problemas difíciles, el algoritmo genético compacto debe ejercer una presión de selección más alta. Esto a su vez, incrementa la probabilidad de supervivencia de bloques constructores de orden superior. En otras palabras, la presión de selección puede simular el papel de una memoria [2].

Aunque es posible aumentar la presión de selección mediante el uso de torneos de mayor tamaño, este enfoque requiere del uso de memoria adicional proporcional al tamaño del torneo.

Con el fin de resolver problemas de optimización difíciles sin comprometer la memoria ni el costo computacional requerido, Ahn y Ramakrishna [2] propusieron el algoritmo genético compacto con elitismo persistente (pe-cGa) y el algoritmo genético compacto con elitismo no persistente (ne-cGa). El elitismo puede aumentar la presión de selección al prevenir la pérdida de los genes con poca prominencia debido a una selección con presión deficiente [2]. Sin embargo, el grado de elitismo debe ser ajustado de forma propia y cautelosa, dado que una alta presión de selección puede causar convergencia prematura.

Algoritmo 10 Algoritmo genético compacto

Entrada: n : tamaño de población, l : longitud del cromosoma
1) Inicializar al vector de probabilidades
Para $i := 1$ **hasta** l
$p[i] := 0.5$
Fin Para
2) Generar dos individuos por medio del vector de probabilidades
a := GenerarIndividuo(p)
b := GenerarIndividuo(p)
3) Realizar competencia entre los individuos
ganador, perdedor := competencia(a, b)
4) Actualizar el vector de probabilidades
Para $i := 1$ **hasta** l
Si ganador$[i] \neq$ perdedor$[i]$ **entonces**
Si ganador$[i] = 1$ **entonces**
$p[i] := p[i] + 1/n$
Si no
$p[i] := p[i] - 1/n$
Fin Si
Fin Si
Fin Para
5) Comprobar convergencia
Para $i := 1$ **hasta** l
Si $p[i] > 0$ && $p[i] < 1$ **entonces**
Ir al paso 2
Fin Si
Fin Para
6) Construir la solución final
Para $i := 1$ **hasta** l
Si $p[i] <= 0$ **entonces**
$p[i] := 0$
Si no, si $p[i] >= 1$ **entonces**
$p[i] := 1$
Fin Si
Fin Para
7) La solución final es p

La forma en que se incorpora el elitismo sobre el algoritmo genético compacto es la siguiente. De los dos individuos a competir, sólo el perdedor es reemplazado por un nuevo individuo, por lo que el ganador nunca es eliminado, sino que solamente es reemplazado hasta encontrar a un individuo con mejor aptitud. A este esquema se le llama algoritmo genético compacto con elitismo persistente. Las modificaciones se pueden observar en el algoritmo 11.

Por otro lado, un elitismo fuerte puede causar una convergencia prematura (a soluciones sub-óptimas). Esto se debe a que las soluciones alcanzan un equilibrio muy temprano gracias al uso de elitismo y a una pérdida de diversidad genética. Por lo tanto, se introduce un nuevo parámetro η. Este nuevo parámetro restringe el número de generaciones que el individuo elitista puede permanecer, provocando una recuperación en la diversidad genética. A este esquema se le llama algoritmo genético compacto con elitismo no persistente. Los cambios requeridos para incluir el elitismo no persistente se muestran en el algoritmo 12. Finalmente, los autores demuestran que el algoritmo genético compacto con elitismo persistente es equivalente a la estrategia evolutiva $(1 + 1) - ES$ con mutación auto-adaptativa.

Algoritmo 11 Algoritmo genético compacto con elitismo persistente

Entrada: n : tamaño de población, l : longitud del cromosoma

1) Inicializar al vector de probabilidades
Para $i := 1$ **hasta** l
$p[i] := 0.5$
Fin Para
Si Es la primera generación **entonces**
2) Generar dos individuos por medio del vector de probabilidades
N := GenerarIndividuo(p)
E := GenerarIndividuo(p)
Si no
2) Generar un nuevo individuo por medio del vector de probabilidades
N := GenerarIndividuo(p)
Fin Si
3) Realizar competencia entre el nuevo individuo y el individuo elitista
ganador, perdedor := competencia(a, b)
E := ganador
4) Actualizar el vector de probabilidades
Para $i := 1$ **hasta** l
Si ganador$[i] \neq$ perdedor$[i]$ **entonces**
Si ganador$[i] = 1$ **entonces**
$p[i] := p[i] + 1/n$
Si no
$p[i] := p[i] - 1/n$
Fin Si
Fin Si
Fin Para
5) Verificar criterio de paro
Para $i := 1$ **hasta** l
Si $p[i] > 0$ && $p[i] < 1$ **entonces**
Ir al paso 2
Fin Si
Fin Para
6) Construir la solución final
Para $i := 1$ **hasta** l
Si $p[i] <= 0$ **entonces**
$p[i] := 0$
Si no, si $p[i] >= 1$ **entonces**
$p[i] := 1$
Fin Si
Fin Para
7) La solución final es p

Algoritmo 12 Algoritmo genético compacto con elitismo no persistente

Entrada: Reemplazar los pasos 2 y 3 del algoritmo 11

Si Es la primera generación **entonces**

2) Generar dos individuos por medio del vector de probabilidades

θ := 0

N := GenerarIndividuo(p)

E := GenerarIndividuo(p)

Si no

2) Generar un nuevo individuo por medio del vector de probabilidades

N := GenerarIndividuo(p)

Fin Si

3) Realizar competencia entre el nuevo individuo y el individuo elitista

ganador, perdedor := competencia(N, E)

Si $\theta \leq \eta$ && ganador $= E$ **entonces**

E := ganador

$\theta := \theta + 1$

Si no

E := GenerarIndividuo(p)

$\theta := 0$

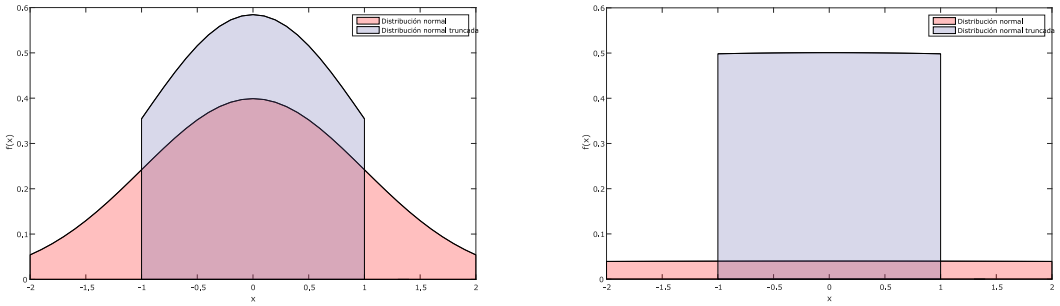
Fin Si

4.3. Real-Valued Compact Genetic Algorithm

El algoritmo genético compacto de valor real fue propuesto por Mininno et al. [37] con el objetivo de resolver problemas de diseño de sistemas de control en línea en un microcontrolador de punto flotante. Los autores argumentan que la implementación de un algoritmo genético compacto no es trivial dado que la mayoría de las plataformas de los microcontroladores utilizan lenguajes orientados a objetos y variables de punto flotante. Adicionalmente, el costo de codificación y decodificación de las variables contrarresta la ventaja principal de un algoritmo genético compacto, que es la reducción de la memoria requerida.

Para resolver este problema, propusieron una variante al algoritmo genético compacto que trabaja de forma directa con variables de punto flotante llamado algoritmo genético compacto de valor real (rcGA, por sus siglas en inglés).

Para un problema de optimización de n variables, el rcGA utiliza como el vector de probabilidades una matriz de tamaño $n \times 2$, que describe la media y la desviación estándar de la distribución de cada gen en la población hipotética. Se asume que la distribución asociada a cada gen del cromosoma será descrita mediante una función de distribución de probabilidad normal truncada en el intervalo de -1 a 1 y por lo tanto se asume que todos los valores reales en el cromosoma (genes) también están normalizados en el mismo intervalo. Las figuras 4-1a y 4-1b muestran los efectos de la normalización de la distribución normal con media 0 y desviación estándar 1 y 10, respectivamente.



(a) media 0 y desviación estándar 1 (b) media 0 y desviación estándar 10

Figura 4-1: Ejemplo de una distribución normal y una distribución normal truncada en el intervalo $[-1, 1]$

La función de distribución de probabilidad truncada en el intervalo $[-1, 1]$ tiene la forma:

$$\text{PDF}(x, \mu, \sigma) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}} \sqrt{\frac{2}{\pi}}}{\sigma \left(\text{erf}(\frac{\mu+1}{\sqrt{2}\sigma}) - \text{erf}(\frac{\mu-1}{\sqrt{2}\sigma}) \right)} \tag{4.1}$$

donde erf es la función de error y se define de la siguiente forma:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \text{d}t \tag{4.2}$$

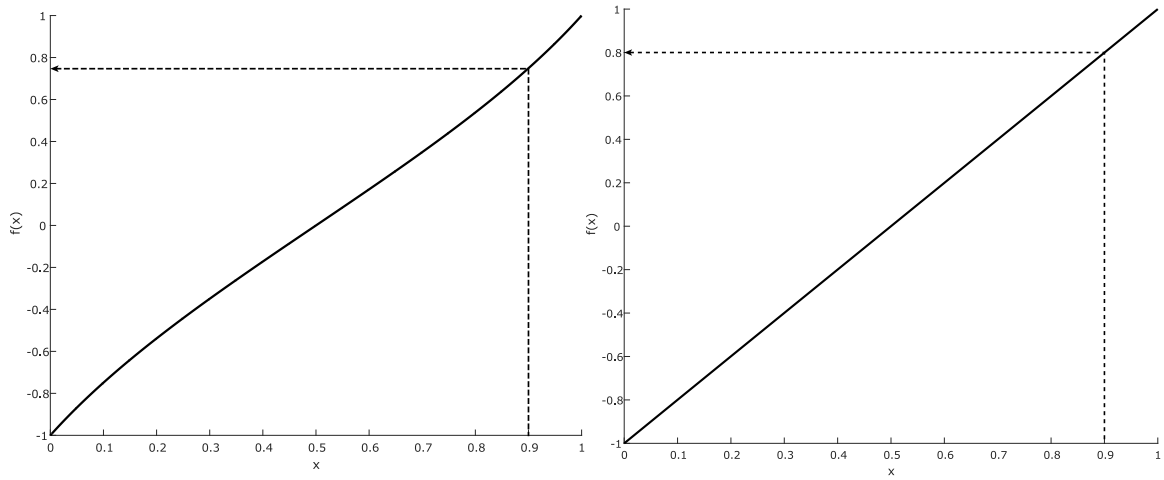
La función de distribución acumulada para esta función de distribución de probabilidad, es aproximada mediante polinomios de Chebyshev de acuerdo al procedimiento descrito en [5].

La estructura general del rcGA es muy similar a la del algoritmo genético compacto con elitismo no persistente. Sin embargo, tanto el mecanismo para generar nuevos individuos como el mecanismo para actualizar al vector de probabilidades son modificados para utilizar valores reales.

Inicialmente, se establece la media en cero y la desviación estándar en un valor suficientemente grande que los autores denominan como λ (generalmente $\lambda := 10$). Con el fin de simular una distribución uniforme en el inicio, ver la figura 4-1b.

4.3.1. Generación de nuevos individuos

La función de distribución acumulada truncada en el intervalo $[-1, 1]$ tiene la forma: $F : [-1, 1] \rightarrow [0, 1]$, por lo que al calcular el inverso de dicha función, se obtiene la función $F^{-1} : [0, 1] \rightarrow [-1, 1]$, en donde es posible aplicar un número aleatorio uniforme entre cero y uno en su dominio, para obtener un número entre menos uno y uno. Esto se puede observar en las figuras 4-1a y 4-1b donde el valor del aleatorio para ambos casos es 0.9 obteniendo un valor de 0.7490 para una media de cero y desviación estándar uno y 0.7995 para una media de cero y desviación estándar diez. La generación de un nuevo individuo, es mediante la aplicación de este proceso utilizando los valores del vector de probabilidades $\mu_i, \sigma_i, \forall i \in [1, n]$.



(a) media 0 y desviación estándar 1

(b) media 0 y desviación estándar 10

Figura 4-2: Ejemplo de la generación de un número con distribución uniforme truncada en el intervalo $[0, 1]$ por medio de un aleatorio entre cero y uno utilizando su función de distribución acumulada inversa.

4.3.2. Actualización del vector de probabilidades

Para decidir la dirección donde se ajustará la nueva media, se realiza un torneo binario entre el individuo elitista y el nuevo individuo generado. Con base en el resultado del torneo, se moverá la nueva media de forma que se acerque al ganador con un tamaño de paso proporcional al número de elementos de la población hipotética. A continuación, se muestran las fórmulas para actualizar al vector de probabilidades y la estructura completa del rcGA el cual se puede ver en el algoritmo 13.

$$\mu^{k+1} = \mu^k + \frac{1}{n}(\text{ganador} - \text{perdedor}) \tag{4.3}$$

$$\sigma^{(k+1)^2} = \sigma^{(k)^2} + \mu^{(k)^2} - \mu^{(k+1)^2} + \frac{1}{n}(\text{ganador}^2 - \text{perdedor}^2) \tag{4.4}$$

Algoritmo 13 Algoritmo genético compacto de valor real

Entrada: n: Tamaño de la población, λ: desviación estándar inicial, η: el número máximo de generaciones que puede sobrevivir el individuo elitista, m: número de variables del problema

1) Inicialización del vector de probabilidades

Para $i := 1$ **hasta** m

$\mu_i := 0$

$\sigma_i := \lambda$

Fin Para

2) Generar un cromosoma por medio del vector de probabilidades

Si es la primera generación **entonces**

$\theta := 0$

$E :=$ GenerarIndividuo(μ, σ)

Fin Si

$N :=$ GenerarIndividuo(μ, σ)

3) Realizar un torneo entre el individuo elitista y el último individuo generado

ganador, perdedor := competencia(E, N)

Si $\theta < \eta$ && $ganador = E$ **entonces**

$\theta := \theta + 1$

Si no, si ganador $= N$ **entonces**

$E :=$ ganador

$\theta := 0$

Si no

$E :=$ GenerarIndividuo(μ, σ)

$\theta := 0$

Fin Si

4) Actualizar el vector de probabilidades

Para $i := 1$ **hasta** m

$\mu_i^{k+1} := \mu_i^k + \frac{1}{n}(\text{ganador} - \text{perdedor})$

$\sigma_i^{(k+1)^2} := \sigma_i^{(k)^2} + \mu_i^{(k)^2} - \mu_i^{(k+1)^2} + \frac{1}{n}(\text{ganador}^2 - \text{perdedor}^2)$

Fin Para

5) Incrementar el número de iteraciones e ir al paso 2 si el críterio de terminación aún no se cumple

4.4. Optimizador mediante cúmulos de partículas compacto

Propuesto por Neri et al. [39] en octubre del 2012. Diseñado principalmente para su uso en aplicaciones donde las especificaciones de hardware son limitadas. Esta metaheurística utiliza la lógica de búsqueda de un PSO típico, pero utiliza un vector de probabilidades para representar al cúmulo. Los autores argumentan que el consumo de memoria de esta metaheurística es equivalente a almacenar únicamente cinco partículas.

El optimizador por cúmulos de partículas compacto (cPSO), es una modificación del algoritmo genético compacto de valor real. Por lo tanto, utiliza un vector de probabilidades representado mediante una matriz de tamaño $n \times 2$, una función de distribución normal truncada para representar las posición del cúmulo hipotético y se asume que todas las variables están normalizadas en el intervalo $[-1, 1]$. Las fórmulas utilizadas para actualizar al vector de probabilidades son las mismas que utiliza rcGA, que fueron presentadas en las ecuaciones(4.3) y(4.4).

La idea general del cPSO es la misma a la utilizada por rcGA: se realiza un torneo entre dos individuos para realizar una ligera modificación al vector de probabilidades. Sin embargo, a diferencia del rcGA, el cPSO genera únicamente a un individuo mediante el vector de probabilidades y el otro individuo es generado por medio de las fórmulas para actualizar la posición de una partícula. El individuo generado mediante el vector de probabilidades, es utilizado como el mejor registro histórico de la partícula ($lbest$), de tal forma que influya al movimiento de la partícula hacia nuevas posiciones del espacio de búsqueda mientras también intenta moverse en dirección al mejor registro global del cúmulo hipotético ($gbest$).

Los parámetros utilizados por el cPSO fueron optimizados mediante un PSO con población utilizando como solución inicial las recomendaciones propuestas en [41]. De forma más específica, se utilizó como valor de la constante de inercia $w = -0.2$, como factor de aprendizaje social $c1 = 3.74$ y como factor de aprendizaje cognitivo $c2 = -0.07$. El algoritmo 14 muestra la estructura del PSO compacto. Finalmente, cabe destacar que los autores argumentan que en los resultados de su estudio, observaron que en general el

Algoritmo 14 Optimizador mediante cúmulos de partículas compacto

Entrada: n: Tamaño de la población hipotética, λ: desviación estándar inicial, m: número de variables del problema

1) Inicialización del vector de probabilidades

Para $i := 1$ **hasta** m

$\mu_i := 0$

$\sigma_i := \lambda$

Fin Para

2) Inicializar de forma aleatoria, x, v **y** $gbest$

Mientras criterio de terminación no se cumpla

3) Generar un nuevo individuo por medio del vector de probabilidades

$lbest :=$ GenerarIndividuo(μ, σ)

4) Actualizar la posición y velocidad de la partícula

$v := v + c_1 * rand(0,1) \times (gbest - x) + c_2 * rand(0,1) \times (lbest - x)$

$x := x + v$

5) Realizar un torneo entre el nuevo individuo generado y la nueva posición de la partícula

ganador, perdedor := competencia($x, lbest$)

6) Actualizar al vector de probabilidades

Para $i := 1$ **hasta** m

$\mu_i^{k+1} := \mu_i^k + \frac{1}{n}(\text{ganador} - \text{perdedor})$

$\sigma_i^{(k+1)^2} := \sigma_i^{(k)^2} + \mu_i^{(k)^2} - \mu_i^{(k+1)^2} + \frac{1}{n}(\text{ganador}^2 - \text{perdedor}^2)$

Fin Para

7) Actualizar al mejor histórico

ganador, perdedor := competencia($x, gbest$)

$gbest :=$ ganador

Fin Mientras

costo computacional de una metaheurística compacta es mayor que el costo en su versión poblacional y además descubrieron que la exploración de un espacio dimensional alto, es especialmente difícil para las metaheurísticas compactas debido a que todas las soluciones son obtenidas a partir de un mismo vector de probabilidades.

4.5. Evolución diferencial compacta

La evolución diferencial compacta (cDE, por sus siglas en inglés), fue propuesta por Mininno et al. [36] en noviembre del 2009. Al igual que las demás metaheurísticas compactas, utiliza un vector de probabilidades para describir la distribución de soluciones en una población hipotética. La evolución diferencial compacta reproduce la lógica de búsqueda de la evolución diferencial [52] al utilizar los operadores de cruza y mutación para generar nuevos individuos.

El algoritmo de la evolución diferencial consiste en los siguientes pasos. Generar una población aleatoria de n individuos mediante una distribución uniforme. En cada generación y para cada individuo x_i de la población, seleccionar a tres individuos x_r, x_s y x_t en la población de forma aleatoria. Producir a un nuevo individuo x' por medio de la mutación. Específicamente, $x' = x_t + F(x_r - x_s)$, donde $F \in [0, 2]$ es una constante que controla la longitud de la exploración. Finalmente, se obtendrá al individuo final mediante la cruza, que consiste en decidir, mediante una probabilidad, para cada gen del cromosoma, si el gen a seleccionar será del individuo x_k o será del nuevo individuo obtenido mediante mutación.

El esquema de mutación anteriormente mencionado es el más utilizado. Sin embargo, existe una gran variedad de esquemas de mutación. A continuación se presentan algunos de ellos. Para obtener una descripción detallada sobre los esquemas y sus aplicaciones, se recomienda al lector consultar [9].

1. DE/best/1: $x' = x_{best} + F(x_r - x_s)$
2. DE/current-to-best/1: $x' = x_k + F(x_{best} - x_k) + F(x_r - x_s)$
3. DE/best/2: $x' = x_{best} + F(x_r - x_s) + F(x_u - x_v)$

4. DE/rand/2: $x' = x_t + F(x_r - x_s) + F(x_u - x_v)$

5. DE/rand-to-best/1: $x' = x_t + F(x_{best} - x_t) + F(x_r - x_s)$

6. DE/rand-to-best/2: $x' = x_t + F(x_{best} - x_t) + F(x_r - x_s) + (x_u - x_v)$

La selección del superviviente consiste en seleccionar al mejor individuo mediante una comparación entre el individuo x_k y el nuevo individuo generado mediante mutación y cruza x'. Este enfoque de selección es muy similar al torneo binario utilizado por el algoritmo genético compacto para actualizar al vector de probabilidades.
La forma en que adaptaron la lógica de la evolución diferencial para realizar la evolución diferencial compacta, fue directa. Se utilizaron las mismas fórmulas empleadas por el algoritmo genético compacto de valor real para actualizar al vector de probabilidad. Por lo tanto, se asume que todas las variables están normalizadas en el intervalo $[-1, 1]$, y se asume que la distribución de los genes en la población es representada mediante una función de distribución normal truncada. Al inicio, se establecen todos los valores del vector de probabilidades con media 0 y desviación estándar 10, de tal forma que se represente una distribución uniforme de los genes en la población inicial. El proceso evolutivo es muy similar al de la evolución diferencial. Inicialmente, se genera un individuo por medio del vector de probabilidades que tomará el papel del individuo elitista. En cada generación, se producen tres individuos x_r, x_s y x_t por medio del vector de probabilidades. Se aplica el operador de mutación anteriormente descrito para generar un nuevo individuo. Posteriormente, se realiza un torneo entre el nuevo individuo y el individuo elitista. Con base en el resultado, se modificará al vector de probabilidades de tal forma que se acerque al ganador con un tamaño de paso proporcional al número de elementos en la población hipotética. La estructura de la evolución diferencial compacta se puede ver en el algoritmo 15.

4.6. Evolución diferencial compacta multi-objetivo

En diciembre del 2014, Osorio et al. [53] propusieron una metaheurística compacta para problemas de optimización multi-objetivo basada en la evolución diferencial compacta llamada evolución diferencial compacta multi-objetivo (mocDE).

Algoritmo 15 Evolución diferencial compacta

Entrada: n: Tamaño de la población hipotética, λ: desviación estándar inicial, m: número de variables del problema, $F \in [0, 2]$: longitud exploración, P_c: probabilidad de cruza

1) Inicialización del vector de probabilidades
Para $i := 1$ **hasta** m
$\mu_i := 0$
$\sigma_i := \lambda$
Fin Para
2) Generar por medio del vector de probabilidades al individuo elitista
$E :=$ GenerarIndividuo(μ, σ)
Mientras criterio de terminación no se cumpla
3) Generar tres individuos por medio del vector de probabilidades
$x_r :=$ GenerarIndividuo(μ, σ)
$x_s :=$ GenerarIndividuo(μ, σ)
$x_t :=$ GenerarIndividuo(μ, σ)
4) Aplicar el operador de mutación sobre éstos individuos
$x := x_t + F\,(x_r - x_s)$
5) Aplicar el operador de cruza sobre el individuo elitista y el nuevo individuo
Para $i := 1$ **hasta** m
Si aleatorio$(0, 1) > C_r$ **entonces**
$x_i := E_i$
Fin Si
Fin Para
6) Realizar un torneo entre el nuevo individuo y el individuo elitista
ganador, perdedor := competencia(x, E)
7) Actualizar al vector de probabilidades
Para $i := 1$ **hasta** m
$\mu_i^{k+1} := \mu_i^k + \frac{1}{n}(\text{ganador} - \text{perdedor})$
$\sigma_i^{(k+1)^2} := \sigma_i^{(k)^2} + \mu_i^{(k)^2} - \mu_i^{(k+1)^2} + \frac{1}{n}(\text{ganador}^2 - \text{perdedor}^2)$
Fin Para
8) Selección del nuevo individuo elitista
ganador, perdedor := competencia(x, E)
$E :=$ ganador
Fin Mientras

Al igual que las demás metaheurísticas compactas, utiliza un vector de probabilidades para representar a una población de soluciones. Se asume que la distribución de los genes es representada mediante una función de distribución normal truncada en el intervalo $[-1, 1]$. Por lo tanto, también se asume que todas las variables están normalizadas en dicho intervalo.

Al inicio, se establecen todos los valores del vector de probabilidades con media 0 y desviación estándar 10, de forma que el vector de probabilidades represente a una población distribuida uniformemente sobre el espacio de búsqueda. Inicialmente, se genera a un individuo por medio del vector de probabilidades que servirá para guiar el proceso de optimización. Este nuevo individuo es almacenado en todas las posiciones del archivo de soluciones y también es utilizado para establecer los valores iniciales del vector de referencia. En cada iteración se genera un nuevo individuo por medio de la mutación y cruza de la evolución diferencial. Este nuevo individuo tendrá que competir contra el individuo elitista mediante dominancia de Pareto. Si el nuevo individuo domina al individuo elitista o el tiempo de vida del individuo elitista se agota, el nuevo individuo es almacenado en el archivo y toma el papel del individuo elitista. En caso de ser incomparables entre sí, se intentará agregar al nuevo individuo en el archivo. En caso de ser agregado, el nuevo individuo tomará el papel de individuo elitista. De lo contrario, sólo se incrementa la edad del individuo elitista. Finalmente, se actualizan los valores del vector de probabilidades, de forma que éste se acerque en proporción al tamaño de población al valor del individuo elitista. Las fórmulas para actualizar al vector de probabilidades son las fórmulas utilizadas por el algoritmo genético compacto de valor real definidas en las ecuaciones (4.3) y (4.4).

4.6.1. Algoritmo para almacenar soluciones no dominadas

Lo último que falta por discutir, es, ¿cómo se almacenan las soluciones no dominadas en el archivo de soluciones? El procedimiento utilizado por mocDE es el siguiente. Para una nueva solución x que se intenta almacenar en el archivo, se asume que x domina al individuo elitista o que x no es dominada por el individuo elitista.

Se asume que el archivo tiene un tamaño p y cada posición del archivo está asociada con

Algoritmo 16 Evolución diferencial compacta multi-objetivo

Entrada: n: número de variables del problema, p: tamaño de la población hipotética, α: desviación estándar inicial, m: número de objetivos del problema, $F \in [0, 2]$: longitud de la exploración, C_r: probabilidad de cruza, η: número máximo de generaciones que puede sobrevivir el individuo elitista, λ: Conjunto de vectores de ponderación

Salida: A: archivo que almacena las soluciones no dominadas

1) Inicialización del vector de probabilidades
Para $i := 1$ **hasta** m
$\mu_i := 0$
$\sigma_i := \alpha$
Fin Para
2) Generar por medio del vector de probabilidades al individuo elitista
$e :=$ GenerarIndividuo(μ, σ)
$\theta := 0$
3) Establecer el punto de referencia como el individuo elitista $z := e$
4) Almacenar todas al individuo elitista en el archivo
Mientras criterio de terminación no se cumpla
5) Generar tres individuos x_r, x_s **y** x_t **por medio del vector de probabilidades**
6) Aplicar el operador de mutación sobre éstos individuos
$x := x_t + F\,(x_r - x_s) + F\,(e - x_t)$
7) Aplicar el operador de cruza sobre el individuo elitista y el nuevo individuo
Para $i := 1$ **hasta** m
Si aleatorio$(0, 1) > C_r$ **entonces**
$x_i := e_i$
Fin Si
Fin Para
8) Realizar un torneo entre el nuevo individuo y el individuo elitista
Si $f(x) \prec f(e) \vee \theta \geq \eta$ **entonces**
ganador $:= x$, perdedor $:= e$ $e := x$ $\theta := 0$
Si no, si $f(e) \not\prec f(x) \wedge$ archivo(A, z, x, λ) **entonces**
ganador $:= x$ perdedor $:= e$ $e := x$ $\theta := 0$
Si no
ganador $:= e$ perdedor $:= x$ $\theta := \theta + 1$
Fin Si
9) Actualizar al vector de probabilidades
Para $i := 1$ **hasta** m
$\mu_i^{k+1} := \mu_i^k + \frac{1}{n}(\text{ganador} - \text{perdedor})$
$\sigma_i^{(k+1)^2} := \sigma_i^{(k)^2} + \mu_i^{(k)^2} - \mu_i^{(k+1)^2} + \frac{1}{n}(\text{ganador}^2 - \text{perdedor}^2)$
Fin Para
Fin Mientras

un subproblema de ponderación de Tchebycheff. Por lo tanto, la nueva solución x será comparada respecto a todas las soluciones del archivo utilizando el vector de ponderación correspondiente al subproblema a optimizar. En el caso de que el valor de la nueva solución al evaluar la métrica ponderada de Tchebycheff supere al valor de la solución actualmente almacenada, entonces la nueva solución reemplaza a la solución anterior. El algoritmo 17 muestra la forma en que se almacenan las soluciones no dominadas.

Algoritmo 17 Función para almacenar soluciones no dominadas

Entrada: λ: conjunto de vectores de ponderación, p: número de subproblemas de ponderación, A: archivo, z: punto de referencia, k: número de objetivos, x: solución candidata a ser almacenada
Salida: un booleano que indica si la solución fue almacenada

1) Actualizar el valor del punto de referencia
Para $i := 1$ **hasta** k
  Si $f_i(x) < z_i$ **entonces**
    $z_i := x_i$
  Fin Si
Fin Para
$agregada := falso$
2) Intentar almacenar la solución en cada subproblema
Para $i := 1$ **hasta** p
  $f_1 := \text{máx}_{1 \leq j \leq k} \lambda_{i,j} |f_j(x) - z_j|$
  $f_2 := \text{máx}_{1 \leq j \leq k} \lambda_{i,j} |f_j(A_i) - z_j|$
  Si $f_1 < f_2$ **entonces**
    $A_i := x$
    $agregada := verdadero$
  Fin Si
Fin Para
Regresar $agregada$

Capítulo

5

Optimización multi-objetivo usando un algoritmo compacto de cúmulos de partículas

En este capítulo se presenta una nueva metaheurística compacta basada en PSO para problemas de optimización multi-objetivo continuos y sin restricciones llamada *compact Multi-Objective Particle Swarm Optimizer* ($compactMOPSO$). Esta metaheurística utiliza el enfoque de descomposición propuesto por Osorio et al. en [53], que consiste en descomponer un problema de optimización multi-objetivo en una serie de subproblemas de métricas ponderadas y resolverlos de forma simultánea. Las soluciones no dominadas obtenidas durante el proceso de búsqueda son almacenadas en un archivo externo, donde cada posición del archivo está asociada con uno de los subproblemas. Por lo tanto, para que una solución sea almacenada en una determinada posición del archivo, ésta deberá resolver de mejor forma el subproblema que la solución almacenada en ese momento.

Para resolver cada uno de los subproblemas, se utiliza un nuevo PSO compacto para problemas mono-objetivo. El diseño de esta nueva metaheurística es inspirado tanto en el algoritmo genético compacto con elitismo persistente [2] como en el optimizador por cúmulos de partículas compacto propuesto por Neri et al. [39]. Ambas metaheurísticas son descritas en el capítulo anterior.

La primera sección pretende explicar de forma breve las ventajas que ofrece el uso de

una población en un algoritmo evolutivo. El conocer estas ventajas puede ser de utilidad al momento de diseñar una metaheurística compacta. En la segunda sección se presenta nuestra propuesta de un PSO compacto para problemas mono-objetivo. Las secciones posteriores están enfocadas a explicar cómo se extendió el PSO compacto con el fin de resolver problemas de optimización multi-objetivo. Inicialmente se describe la estructura general de la metaheurística, posteriormente se describe el funcionamiento del archivo externo y finalmente se describe la estrategia utilizada para seleccionar al líder.

5.1. Importancia de la población

Aunque es evidente la importancia de la población en los algoritmos evolutivos, no es posible explicar de forma precisa cuál es su contribución dentro de los mismos. Con este fin, Prügel-Bennet [44] realizó un estudio en el que intenta descubrir los casos en los que un algoritmo de búsqueda con población tiene ventajas respecto a uno sin población y de esta forma distinguir los beneficios que ofrece una población. Prügel-Bennet logró identificar cinco casos en los que un algoritmo poblacional tiene ventajas con respecto a un algoritmo no poblacional.

- El primer caso es referente a la diversidad de bloques constructores que se pueden encontrar en una población inicial aleatoria. El autor argumenta que la combinación de los distintos bloques constructores iniciales es la que da lugar a nuevas soluciones prometedoras.

- El segundo caso es referente a algoritmos poblacionales que utilicen un operador de cruza. Al generar un nuevo individuo, si los dos padres comparten un mismo valor sobre una variable, el hijo producto de la cruza conservará el mismo valor. Esto da lugar a búsquedas más efectivas dado que sólo se exploran regiones del espacio de búsqueda en donde los individuos discrepan.

- El tercer caso indica que una población puede ignorar falsos atractores en el paisaje de aptitud. El autor argumenta que esta robustez a los falsos atractores proviene del hecho de que después de realizar un cierto número iteraciones, los individuos de

una población responden a una aptitud promedio. Dicha aptitud promedio de una población ignora el ruido de corto alcance, pero es sensible a tendencias de largo alcance.

- El cuarto caso argumenta que la población puede ser vista como un mecanismo de recuperación en casos donde la posición inicial de un individuo es inconveniente. En otras palabras, dado que un algoritmo poblacional realiza un muestreo múltiple para generar soluciones candidatas, se reduce la probabilidad de que una solución inicial inconveniente pueda comprometer la búsqueda. Además, la población actúa como defensa en caso de que ocurran errores de decisión.

- El quinto caso argumenta que la población puede ser aprovechada para aprender valores de parámetros que pueden ser útiles para balancear la exploración y la explotación. Esto puede ser realizado tanto de forma explícita como implícita. Como ejemplo se menciona que la diversidad en la población controla la forma en que se realiza la búsqueda al utilizar un operador de cruza.

Por estas razones, Neri et al. [39] recomiendan acudir a una metaheurística poblacional siempre que sea posible y acudir a una metaheurística compacta en aplicaciones donde la memoria disponible esté limitada.

5.2. Construcción del algoritmo mono-objetivo

En esta sección se describe la propuesta de un nuevo PSO compacto, que es usado posteriormente para resolver los problemas de métricas ponderadas obtenidos al descomponer el problema multi-objetivo original. Esta propuesta surge con el fin de mejorar el desempeño del PSO compacto descrito en el capítulo anterior, dado que se observó que no tiene un buen desempeño en problemas multi-modales.

5.2.1. Representación de la población

En lugar de generar y almacenar un conjunto de partículas que conformen al cúmulo, se utiliza una representación estadística caracterizada por dos vectores μ y σ, ambos de lon-

gitud n, donde los valores μ_i y σ_i describen en conjunto la distribución de las partículas sobre el espacio de búsqueda en la i-ésima dimensión por medio de una función de distribución de probabilidad normal truncada. En la figura 5-1 se puede observar un ejemplo de la distribución de las partículas sobre un espacio de búsqueda de dos dimensiones. Sin embargo, a diferencia las demás metaheurísticas compactas de valor real (rcGA, cDE, cPSO), la propuesta de esta tesis no trunca el rango de las variables al intervalo $[-1, 1]$, ya que esto requiere de transformar las soluciones del intervalo $[-1, 1]$ al intervalo en que están definidas originalmente cada vez que se desee evaluar a la función objetivo. Por tanto, las variables son truncadas en el mismo intervalo en el que están definidas. La ventaja observada de utilizar una distribución truncada es que se tiene un mayor control al momento de generar soluciones, ya que éstas siempre serán válidas sin importar la desviación estándar utilizada.

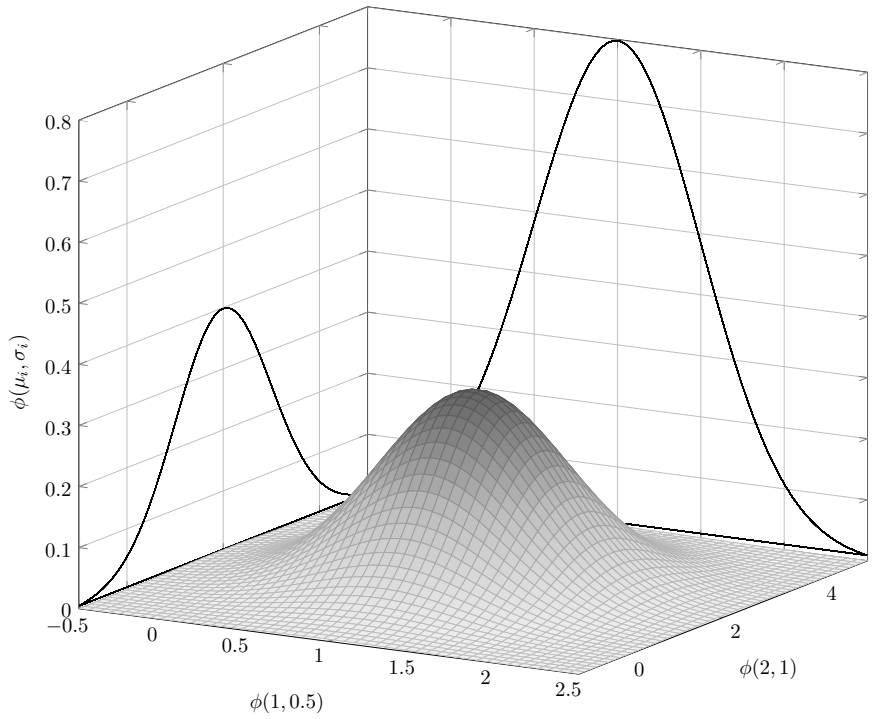


Figura 5-1: En esta figura se ilustra el cúmulo hipotético que es representado por el vector de probabilidades $\mu = [1, 2]$, $\sigma = [0.5, 1]$, donde $\phi(\mu_i, \sigma_i)$ es una función de distribución de probabilidad normal con media μ_i y desviación estándar σ_i.

5.2.2. Construcción de la metaheurística

Como el mismo término *Optimizador mediante cúmulo de partículas* sugiere, éste fue diseñado para utilizarse con más de una partícula. Y de hecho, las fórmulas utilizadas para actualizar la posición y la velocidad de una partícula sólo tienen sentido cuando al menos existen dos partículas. A continuación se muestra un ejemplo que demuestra lo comentado.

Supongamos que se genera de forma aleatoria una sola partícula en el cúmulo con posición x y velocidad inicial v. Entonces, la mejor posición histórica de la partícula ($lbest$) es la posición inicial de la partícula y la mejor partícula del cúmulo es ella misma ($gbest$).

$$v = v + c_1 * rand_i \times (gbest - x) + c_2 * rand_2 \times (lbest - x)$$
$$x = x + v$$

Al sustituir tanto $lbest$ como $gbest$ por x se tiene,

$$v = v + c_1 * rand_i \times (x - x) + c_2 * rand_2 \times (x - x)$$
$$v = v + c_1 * rand_i \times [0, ..., 0]^T + c_2 * rand_2 \times [0, ..., 0]^T$$
$$v = v$$
$$x = x + v$$

Al calcular la nueva velocidad de la partícula, se obtiene nuevamente v; provocando un movimiento constante en la partícula que después de un cierto número de iteraciones terminará fuera de los rangos deseables de búsqueda. Este problema mostrado se debe a que no existe diferencia entre el $lbest$ y el $gbest$. En este caso, la mejor posición de la partícula es también la mejor posición de todo el cúmulo.

Basta con cambiar la definición del $lbest$ o del $gbest$ para resolver dicho problema. En la presente propuesta, $gbest$ tomará el papel de la partícula elitista, es decir, conservará la mejor posición encontrada durante todo el proceso de búsqueda y $lbest$ será utilizado para mover a la partícula hacia nuevas direcciones. En cada iteración se obtiene un nuevo $lbest$ mediante el vector de probabilidades, por lo que en cada iteración, la partícula intentará

desplazarse tanto en la dirección de la mejor solución encontrada como en la dirección indicada por el cúmulo.

Cabe destacar la similitud entre un PSO y el algoritmo genético compacto con elitismo persistente. En ambos, por medio de ciertos operadores se obtiene una nueva solución que intentará reemplazar a la solución elitista. Debido a su similitud con el PSO y dado que el elitismo persistente fue propuesto con el objetivo de aumentar la presión de selección del algoritmo genético compacto para poder resolver problemas de orden superior [2], se usó en este caso como parte de la estructura general del PSO compacto.

5.2.3. Actualización del vector de probabilidades

En esta subsección se discute el mecanismo que utiliza el PSO compacto propuesto para actualizar al vector de probabilidades. Este mecanismo está inspirado en el mecanismo utilizado en [39, 36, 37]. Sin embargo, en esta tesis se realiza un cambio significativo en la forma en que se actualiza el vector.

A continuación, se describe el mecanismo de actualización del vector de probabilidades empleado en el PSO compacto de Neri et al. [39]. En cada iteración se generan dos partículas: una por medio del vector de probabilidades, que toma el papel de la mejor posición local ($lbest$) y la otra por medio de las fórmulas para actualizar la posición de una partícula. Posteriormente, se realiza un torneo binario entre estas partículas generadas con el objetivo de determinar al ganador. Esta información es utilizada para mover al vector de probabilidades hacia la dirección del ganador del torneo, utilizando un tamaño de paso proporcional al efecto que tendría el movimiento de una partícula sobre un cúmulo de tamaño T_c. Sin embargo, la fórmula que los autores proponen para actualizar la media del vector de probabilidades (ver la ecuación 5.1), no siempre dirige a la media hacia la partícula ganadora. En realidad, la media es desplazada en la dirección del vector que va desde la partícula perdedora hacia la ganadora. Esto se puede observar en la figura 5-2 que muestra un caso en el que después de aplicar dicha fórmula, la media se termina alejando de la partícula ganadora aún cuando se encontraba muy cerca de ésta. A partir de lo anterior, queda claro que puede ser perjudicial utilizar al perdedor del torneo para decidir la dirección en la que se moverá la media. Por tanto, es conveniente ignorar al perdedor

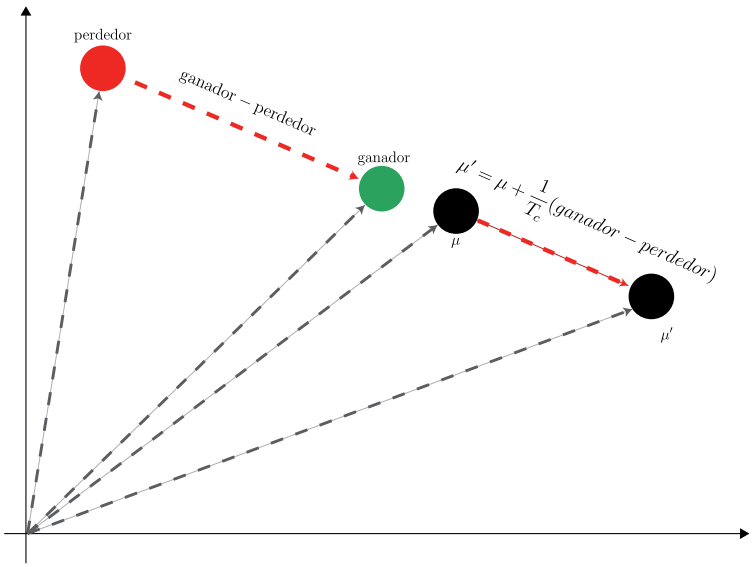


Figura 5-2: Esta figura muestra que la fórmula original puede mover de forma errónea a la media del vector de probabilidades. Supongamos que se realizó un torneo binario entre dos partículas y ya se determinó cuál de ellas es la ganadora y cúal es la perdedora. Por ende, al desplazar la media (μ) en la dirección del vector que va desde el perdedor al ganador, ésta se termina alejando de la partícula ganadora.

del torneo y dirigir directamente a la media en la dirección del ganador. El único caso en el que la media es dirigida rumbo al ganador, es cuando la partícula perdedora está muy cerca de la media o es igual a la media. La ecuación 5.2 presenta la nueva propuesta para actualizar a la media utilizando únicamente la información del ganador del torneo y la figura 5-3 ilustra cómo la fórmula propuesta siempre dirige la media del vector de probabilidades hacia el ganador del torneo.

$$\mu = \mu + \frac{1}{T_c}(\text{ganador} - \text{perdedor}) \tag{5.1}$$

$$\mu = \mu + \frac{1}{T_c}(\text{ganador} - \mu) \tag{5.2}$$

Cálculo de la desviación estándar

Al inicio de la sección se describieron los casos en los que un algoritmo poblacional presenta ventajas respecto a algoritmos no poblacionales. Entre estas ventajas se encuentra la

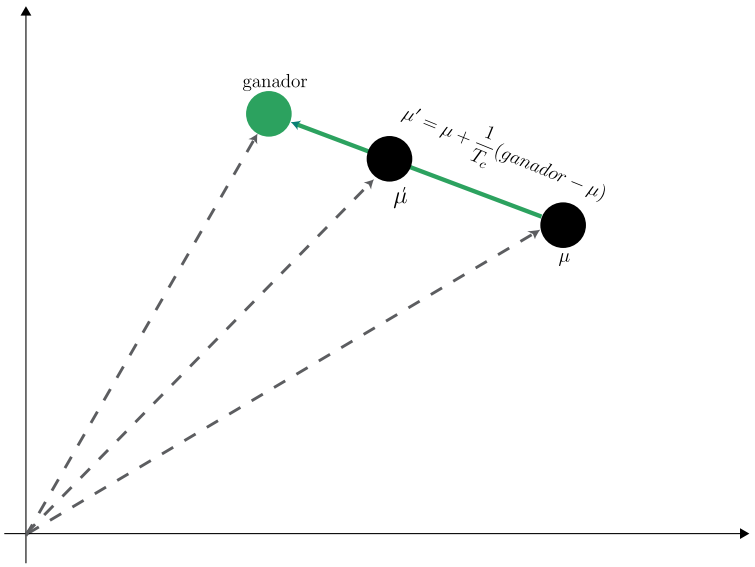


Figura 5-3: Al eliminar la influencia del perdedor del torneo, siempre se dirige a la media en la dirección del ganador.

capacidad de recuperarse de errores de decisión cometidos y la diversidad de soluciones que puede ofrecer una población inicial aleatoria. En nuestra metaheurística se pretende simular dichas características mediante la manipulación de la desviación estándar del vector de probabilidades.

Al inicio del algoritmo, el valor inicial de la desviación estándar es determinado mediante un número muy grande [1]. La razón por la que este número debe ser muy grande, es para simular una distribución uniforme entre los límites de la variables. Supongamos que a es el límite inferior y b el superior. Al utilizar una desviación estándar considerablemente mayor a $|b - a|$ se obtiene una aproximación a una distribución uniforme en el intervalo de $[a, b]$ sin importar cuál sea el valor de la media. De hecho entre más grande sea la desviación estándar, mejor será la aproximación obtenida. Basta con establecer la desviación estándar a $10(b - a)$ para obtener una buena aproximación a una distribución uniforme. Esto se puede observar de forma gráfica en la figura 5-4. De esta forma, es posible obtener al inicio de la ejecución del algoritmo cualquier valor en el intervalo de las variables, lo cual simula a un cúmulo inicial de partículas con posiciones aleatorias en el espacio de

[1]Este valor se recomienda que sea mayor que el rango de búsqueda de las variables. Por ejemplo si el rango válido de una variable x_i es de -5 a 5, la longitud del intervalo es 10 y por lo tanto, el valor inicial de la desviación estándar en este caso debe de superar a 10. Por ejemplo, 100 sería un valor inicial adecuado.

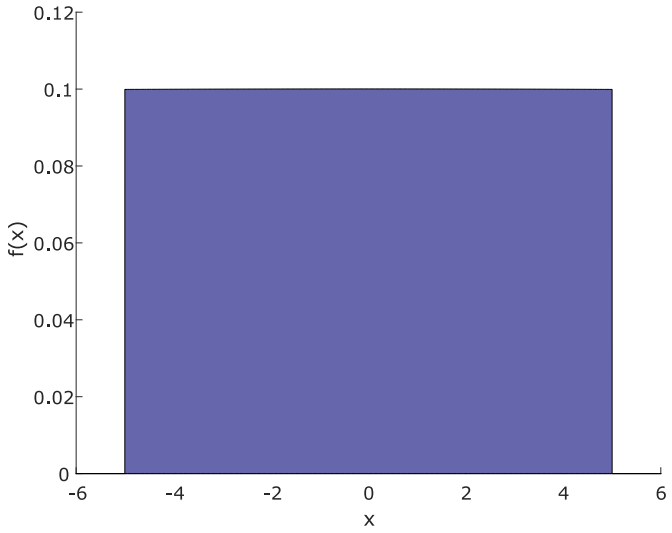


Figura 5-4: Esta figura muestra una distribución normal truncada en el intervalo de -5 a 5 que tiene una desviación estándar 100. Se puede observar claramente que ésta es una buena aproximación a una función de distribución de probabilidad uniforme.

búsqueda.

Posteriormente, se define a la desviación estándar como la distancia entre la media del vector de probabilidades y la partícula elitista ($gbest$). Este mecanismo da la oportunidad a la partícula de corregir su posición en caso de haber tomado una mala decisión. Cuando la media del vector de probabilidades se encuentra relativamente cerca de la partícula elitista, los nuevos individuos generados por medio del vector de probabilidades también lo estarán y cuando la distancia entre la media y la partícula elitista es alta, la distribución normal truncada tenderá a comportase como si fuese una distribución uniforme.

PSO compacto para problemas mono-objetivo

Ahora que fue descrita la forma en que se actualiza al vector de probabilidades, es posible presentar la estructura general de nuestro PSO compacto. El algoritmo 18 presenta un esquema general del PSO compacto propuesto.

Al inicio del algoritmo se generan mediante una distribución aleatoria uniforme a la única partícula (x) del cúmulo y a la partícula elitista ($gbest$). El vector de probabilidades también es inicializado de tal forma que represente una distribucíon uniforme. Esto es logrado haciendo que el valor de las desviaciones estándar de cada variable de decisión sea un

valor muy grande. Se realiza un torneo entre las dos partículas generadas para determinar si deben ser intercambiados en el caso de que x tenga un mejor valor que el de $gbest$ después de ser evaluadas en la función objetivo. A continuación, se describe el proceso iterativo; en cada iteración se obtendrá un nuevo individuo $lbest$ por medio del vector de probabilidades. Este nuevo individuo, junto con $gbest$, son utilizados para calcular la nueva velocidad de la partícula y, posteriormente, afectar su posición. En cada iteración sólo se produce una nueva solución, que resulta ser la partícula desplazada. Dicha solución es comparada con la partícula elitista (al igual que el algoritmo genético compacto con elitismo persistente) y pasará a reemplazarla en el caso que la evaluación de la nueva solución tenga un valor menor que el del $gbest$. A continuación, se actualiza al vector de probabilidades de tal forma que se acerque a la partícula elitista en un tamaño de paso de $1/T_c$. Entre más pequeño sea el valor de $1/T_c$, más tardará el vector de probabilidades en acercarse al $gbest$, dando una mayor oportunidad de recuperarse de algunos errores de decisión cometidos en el proceso de búsqueda. Finalmente, se aplica un mecanismo de mutación[2] para cada variable de decisión. Si un número aleatorio generado con una distribución uniforme entre cero y uno es menor que la probabilidad de aplicar mutación (p_m), la desviación estándar es establecida nuevamente en un valor muy alto, permitiendo que en la próxima iteración, el $lbest$ utilizado para afectar la trayectoria de la partícula, sea generado mediante una distribución uniforme en las variables de decisión en las que se aplicó la mutación.

Cabe destacar que $lbest$ sólo sirve para afectar el movimiento de la partícula y nunca es evaluado en la función. De hecho, sólo se realiza una evaluación de función por iteración, por lo que el número de iteraciones del algoritmo es el número de evaluaciones de la función que se efectúan.

Lo único que queda por comentar, es que se descartó el término de la velocidad en la fórmula para actualizar la velocidad de una partícula, dado que en la mayoría de las ocasiones obstaculizaba su proceso de búsqueda. Por ende, a diferencia del PSO tradicional que sigue tres trayectorias para moverse (su velocidad, su mejor personal y el mejor global), el PSO compacto sólo considera al mejor personal y al mejor global. Esto cambia la

[2] A este operador también se le denomina turbulencia.

Algoritmo 18 Pseudo-código del PSO compacto

Entrada: $F : \mathbb{R}^n \to \mathbb{R}$, n: número de variables del problema, T: número de iteraciones, p_m: probabilidad de mutación, $linf_i$: límite inferior de la variable i, $lsup_i$: límite superior de la variable i, T_c: tamaño del cúmulo hipotético

Salida: $gbest$: mejor resultado encontrado

1 Establecer el valor de c_1 y c_2

$c_1 := c_2 := 2$

2 Inicializar los parámetros del PSO

$v := 0$

Para $i := 1$ hasta n

$x_i :=$ aleatorioUniforme$(linf_i, lsup_i)$

$\mu_i :=$ aleatorioUniforme$(linf_i, lsup_i)$

$\sigma_i := 10(lsup_i - linf_i)$

$gbest_i :=$ aleatorioUniforme$(linf_i, lsup_i)$

Fin Para

3 Actualizar gbest: si $f(x) < f(gbest)$: intercambiar$(x, gbest)$

4 Inicio del proceso iterativo

Mientras $t < (T - 2)$

Para $i := 1$ hasta n

5 Generar lbest por medio de μ, σ

$lbest_i :=$ aleatorioNormal(μ_i, σ_i)

6 Obtener dos números aleatorios

$r1 :=$ aleatorioUniforme$(0, 1)$, $r2 :=$ aleatorioUniforme$(0, 1)$

7 Calcular la velocidad de desplazamiento de la partícula

$v_i := c_1 * r_1 * (gbest_i - x_i) + c_2 * r_2 * (lbest_i - x_i)$

8 Limitar la velocidad al limite inferior o superior, si es el caso

9 Mover la partícula

$x_i := x_i + v_i$

10 Asegurarse que la nueva partícula sea válida; en caso de no serlo, establecer el valor de la partícula al límite inferior o superior, según sea el caso

Fin Para

11 Actualizar al líder del cúmulo: si $f(x) < f(gbest) : gbest := x$

12 Actualizar el vector de probabilidad

Para $i := 1$ hasta n

$\mu_i := \mu_i + \frac{1}{T_c}(gbest_i - \mu_i)$

$\sigma_i := |gbest_i - \mu_i|$

13 Aplicar turbulencia

si aleatorioUniforme$(0, 1) \leq p_m : s_i := 10(lsup_i - linf_i)$

Fin Para

$t := t + 1$

Fin Mientras

interpretación que un PSO compacto puede tener en términos sociales y cognitivos. En un PSO tradicional, una partícula intenta conservar la trayectoria en la que se dirige mientras intenta seguir el éxito de sus compañeros a la vez que quiere seguir la trayectoria que simboliza a su propia experiencia. La partícula de nuestro PSO compacto, por lo general, tiene el deseo de no alejarse de la dirección que representa su mejor éxito ($gbest$). Sin embargo, la partícula intentará seguir la influencia que sus compañeros ejercen sobre ella ($lbest$) y debido a la similitud que tiene con el algoritmo genético con elitismo persistente, siempre dará preferencia a dirigirse rumbo a su mejor éxito. Quizás esta interpretación pueda explicar el porqué puede ser perjudicial el mantener la trayectoria que la partícula traía anteriormente.

5.3. *compactMOPSO*

En la sección anterior, se describió nuestra propuesta de un nuevo PSO compacto para problemas mono-objetivo. Dicha metaheurística es de gran importancia para el desarrollo de este trabajo de tesis debido a que el enfoque utilizado para resolver problemas de optimización multi-objetivo consiste en descomponer el problema original en un conjunto de subproblemas de métricas ponderadas de Tchebycheff y resolver dichos subproblemas de forma simultánea mediante nuestro PSO compacto. Las soluciones de cada uno de los subproblemas de métricas ponderadas de Tchebycheff terminarán conformando el conjunto aproximado de óptimos de Pareto obtenido por nuestro algoritmo.
Debido a que el PSO compacto utiliza una representación estadística de la población, no es posible asignar un subproblema a cada partícula del cúmulo y tampoco es posible almacenar las soluciones no dominadas obtenidas durante el proceso de búsqueda.

5.3.1. Archivo externo

Para poder almacenar las soluciones no dominadas, se utiliza un archivo externo similar al archivo propuesto por Osorio et al. [53] que asigna a cada posición del archivo un vector de ponderación que representa un subproblema de métricas ponderadas de Tchebycheff. Por lo tanto, el tamaño del archivo externo también es el número de subproblemas a resolverse.

El algoritmo 19 muestra cómo se almacena una nueva solución en el archivo externo. Cabe agregar que los vectores de ponderación son generados desde el inicio utilizando el algoritmo descrito en [55].

Algoritmo 19 Algoritmo para almacenar soluciones dentro del archivo externo

Entrada: x: solución candidata a ser almacenada, λ: conjunto de vectores de ponderación, z: vector objetivo ideal, A: archivo, p: número de subproblemas, m: número de objetivos
Salida: Entero positivo que indica la posición del próximo líder a ser utilizado

```
1) Actualizar el vector objetivo ideal
Para i := 1 hasta m
  Si f_i(x) < z_i entonces
    z_i := f_i(x)
  Fin Si
Fin Para
2) Intentar almacenar la solución en cada subproblema
Si archivo está vacío entonces
  Almacenar la solución en todas las posiciones del archivo
  Regresar 1
Si no
  Para i := 1 hasta p
    f_1 := máx_{1≤j≤k} λ_{i,j}|f_j(x) − z_j|
    f_2 := máx_{1≤j≤k} λ_{i,j}|f_j(A_i) − z_j|
    Si f_1 < f_2 y f(A_i) ⊀ f(x) entonces
      A_i := x
    Fin Si
  Fin Para
Fin Si
3) Regresar la posición en el archivo del próximo líder a utilizar
Si se logró agregar la solución al archivo entonces
  Regresar buscarLider(A, λ, z)
Si no
  Regresar aleatorioUniforme(0, p)
Fin Si
```

Cuando se intenta agregar una nueva solución al archivo, primero se consulta el tamaño del archivo. Si el archivo está vacío, la nueva solución será almacenada en todas las posiciones del archivo, ya que inicialmente, la nueva solución es la mejor solución para todos los subproblemas de métricas ponderadas de Tchebycheff. De lo contrario, si el archivo no está vacío, se intentará agregar la nueva solución en todas las posiciones del archivo donde la nueva solución resuelva de mejor forma el problema de métricas ponderadas que

la solución que se encuentra almacenada en la posición correspondiente y que a la vez, la nueva solución no sea dominada por la solución almacenada en el archivo.
Al finalizar el algoritmo, se regresa la posición en el archivo externo del individuo que servirá como el líder del cúmulo en la siguiente iteración. Si la nueva solución fue agregada al archivo, se regresará la posición en el archivo de la solución que tenga el menor valor al resolver el problema de métricas ponderadas asociado a su posición. En otras palabras, si el nuevo individuo es agregado al archivo, el siguiente líder será la solución en el archivo que tenga la menor distancia respecto al vector objetivo ideal. De lo contrario, si no fue agregada la nueva solución al archivo, cualquier solución es apta para ser el líder del cúmulo. Se observó que usar estas dos alternativas para seleccionar al líder permite obtener una mayor diversidad en las posteriores soluciones.

5.3.2. Algoritmo general del *compactMOPSO*

Al igual que su contraparte mono-objetivo, el *compactMOPSO* genera dos soluciones iniciales por medio de una distribución uniforme aleatoria: la única partícula del cúmulo y la partícula que tomará el rol del líder del cúmulo ($gbest$). En caso de que la partícula domine al líder del cúmulo son intercambiados y el líder del cúmulo será almacenado en el archivo. Dado que el archivo se encuentra vacío, éste será almacenado en todas las posiciones del archivo. En cada iteración, se obtiene una nueva solución por medio del vector de probabilidades ($lbest$) y junto con el líder del cúmulo determinarán la nueva velocidad de la partícula x. Posteriormente se moverá la partícula utilizando la nueva velocidad. La partícula en su nueva posición es la nueva solución potencial a entrar al archivo externo. Debido que el *compactMOPSO* preserva un mecanismo similar al elitismo persistente, la nueva solución será comparada mediante dominancia de Pareto con el líder del cúmulo. De la comparación resultan tres casos: si la nueva solución domina al líder o ambos son incomparables, se intentará almacenar a la nueva solución en el archivo externo. De lo contrario, la nueva solución es descartada. Al finalizar el torneo, un nuevo líder es seleccionado utilizando el mecanismo descrito en la subsección anterior. La tarea restante es actualizar al vector de probabilidades y aplicar mutación. Estos mecanismos funcionan de forma similar a los del PSO compacto propuesto. Al terminar el número de iteraciones, se

deben de eliminar las soluciones dominadas del archivo externo. El algoritmo 20 muestra el pseudocódigo general del *compactMOPSO*. Finalmente, cabe destacar que aunque parece que no se utiliza un tamaño de cúmulo hipotético (T_c) para actualizar la media y en su lugar se optó por poner directamente al $gbest$, se está asumiendo que el $T_c = 1$. La razón de esta elección, es que se observó una reducción significativa en el número de evaluaciones de función y una mejora notable en la convergencia conforme se reducía el tamaño del cúmulo hipotético. La forma de actualizar la desviación estándar se tuvo que modificar, ya que al establecer el valor de la media en $gbest$ provoca que la distancia entre la media y $gbest$ sea cero. Para solucionar este problema, se optó por utilizar como la desviación estándar a la distancia entre x y $gbest$.

Algoritmo 20 Pseudo-código del *compactMOPSO*

Entrada: $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$, n: número de variables, m: núm de objetivos, λ: vectores de ponderación, z: vector ideal, A: archivo, p: núm subproblemas T: núm de iteraciones, p_m: probabilidad de mutación, $linf_i$: lím inferior de la variable i, $lsup_i$: lím superior de la variable i

Salida: A: conjunto aproximado de óptimos de Pareto

1 Inicializar el vector objetivo ideal $z_i := \infty \; \forall i \in [1, m]$

2 Inicializar los parámetros del PSO

$c_1 := c_2 := 1.29$

$v := 0$; Inicializar de forma aleatoria a $x, \mu, gbest$ mediante una distribución uniforme. La desviación estándar es inicializada de la misma forma que en su versión monoobjetivo.

3 Obtener $f(x), f(gbest)$**, intercambiar los valores de** x **y** $gbest$ **si** $x \preceq gbest$

4 Agregar $gbest$ **al archivo externo**

5 Inicio del proceso iterativo

Mientras $t < (T - 2)$

Para $i := 1$ hasta n

$lbest_i :=$ aleatorioNormal(μ_i, σ_i)

$r_1 :=$ aleatorioUniforme$(0, 1)$, $r_2 :=$ aleatorioUniforme$(0, 1)$

$v_i := c_1 * r_1 * (gbest_i - x_i) + c_2 * r_2 * (lbest_i - x_i)$

6 Limitar la velocidad al limite inferior o superior, si es el caso

$x_i := x_i + v_i$

7 Asegurarse que la nueva partícula sea válida, en caso de no serlo establecer el valor de la partícula al límite inferior o superior, según sea el caso

Fin Para

8 Selección del nuevo líder:

Si $f(x) \prec f(gbest)$ **entonces**

Si $aleatorioUniforme() < 0.5$ **entonces**

Agregar x **al archivo externo**

Si no

Agregar x **al archivo externo**

Reemplazar al líder del cúmulo

Fin Si

Si no, si $f(gbest) \prec f(x)$ **entonces**

Reemplazar al líder del cúmulo

Si no

Agregar x **al archivo externo**

Reemplazar al líder del cúmulo

Fin Si

Para $i := 1$ hasta n

$\mu_i := gbest_i$, $\sigma_i := |gbest_i - x_i|$

si aleatorioUniforme$(0, 1) \leq p_m : s_i := 10(lsup_i - linf_i)$

Fin Para

$t := t + 1$

Fin mientras

Filtrar las soluciones dominadas en A

Capítulo

6

Estudio experimental

En este capítulo se presenta el estudio experimental realizado para validar el funcionamiento de nuestra propuesta. En la sección 6.1 se describen los problemas de prueba que fueron optimizados tanto por nuestra propuesta como por los algortitmos presentados en el estado del arte. En la sección 6.2 se describen los parámetros utilizados en cada uno de los algoritmos utilizados. La sección 6.3 describe la metodología utilizada para medir y comparar el desempeño de nuestra propuesta respecto a los resultados obtenidos por SMPSO, dMOPSO y mocDE. Finalmente, en la sección 6.4 se realiza la comparación de resultados.

6.1. Problemas de prueba

Con el fin de comparar el desempeño de las distintas metaheurísticas, se ha optado por el uso de distintos conjuntos de funciones de prueba. El primer conjunto a utilizar, es el denominado Ziztler-Deb-Thiele (ZDT), el cual ha sido muy utilizado en el área de optimización evolutiva multi-objetivo, dadas las distintas características que poseen los problemas que pertenecen a éste. Este conjunto de problemas tiene la característica de que que todos sus problemas son escalables respecto al número de variables. Los problemas restantes, son algunos de los problemas de prueba estándar utilizados en la literatura especializada. Para resolver el conjunto de problemas ZDT se utilizó $n = 30$. Para el resto de problemas, se

usó $n = 2$. En específico se usaron los siguientes problemas de prueba:

- ZDT1, ZDT2, ZDT3, ZDT4, ZDT6
- Bihn
- Deb1, Deb2, Deb3
- Fonseca1, Fonseca2
- Laumanns
- Lis
- Murata

Todos los problemas de prueba adoptados tienen dos objetivos.
En el apéndice A, se definen formalmente todos los problemas de prueba utilizados y se describen sus características principales.

6.2. Parámetros utilizados

En la tabla 6.1 se presentan los parámetros partículares de cada algoritmo usado en nuestro estudio comparativo. Se optó por utilizar los parámetros recomendados para cada una de las metaheurísticas, descritos en sus artículos originales.

6.3. Descripción de la metodología utilizada

Los resultados se obtuvieron al realizar 30 ejecuciones independientes de cada uno de los algoritmos, por cada problema de prueba. Se limitó el número de evaluaciones de función a 20,000 y se utilizaron los parámetros individuales descritos en la tabla 6.1. Con el objetivo de comparar los resultados, se hace uso de tres indicadores de desempeño: el indicador de hipervolumen [58], el indicador de espaciado [45], y la distancia generacional invertida más [27]. Éstos son definidos en el apéndice B. Finalmente, para confirmar los resultados obtenidos, se adoptó la prueba de suma de rangos Wilcoxon de dos colas, utilizando un

Parámetros	compactMOPSO	SMPSO	dMOPSO	mocDE
Tamaño de población	1	100	100	1
Número iteraciones	20,000	200	200	20,000
Tamaño del archivo	100	100	100	100
Número de evaluaciones	20,000	20,000	20,000	20,000
W	-	0.1	$U(0.1, 0.5)$	-
$C1$	1.29	$U(1.5, 2.5)$	$U(1.2, 2.0)$	-
$C2$	1.29	$U(1.5, 2.5)$	$U(1.2, 2.0)$	-
T_A	-	-	2	-
Variación diferencial F	-	-	-	1
Probabilidad de cruza P_c	-	-	-	0.1
Probabilidad de mutación P_m	0.01	$1/$número variables	-	-
Supervivencia máxima	-	-	-	0

Tabla 6.1: Parámetros utilizados al realizar este experimento para cada algoritmo.

nivel significativo del $5\,\%$. Todos los algoritmos fueron ejecutados en el mismo equipo, que cuenta con un procesador Intel(R) core i7 con una frecuencia de reloj de 2.9GHz y 32GB de memoria RAM.

Para determinar la capacidad de nuestra propuesta, se dividió el estudio experimental en dos fases. La primera fase consistió en comparar el desempeño de nuestra propuesta con respecto al de dos MOPSOs del estado del arte: SMPSO [38] y dMOPSO [54]. La segunda fase consistió en comparar el desempeño de nuestra propuesta respecto al del único AEMO compacto existente hasta la fecha: mocDE [53].

Para poder establecer una jerarquía sobre los algoritmos, se decidió utilizar un sistema de puntajes, que consiste en darle la mayor puntuación al primer lugar, y la menor al último lugar y finalmente sumar los puntos obtenidos por cada algoritmo. El algoritmo con mayor puntaje obtendrá el primer lugar, y así sucesivamente con los restantes. Esta forma de determinar el orden de los algoritmos nos permite observar el desempeño general de los algoritmos. Como ejemplo, supongamos tres algoritmos A, B y C, y supongamos también que de 5 pruebas, el algoritmo C, se mantuvo en segundo lugar las 5 ocasiones, y que el algoritmo A en 4 ocasiones obtuvo el primer lugar. Por lo tanto, el algoritmo B obtuvo 1 primer lugar y 4 terceros lugares. Entonces, se espera que el algoritmo C obtenga un segundo lugar, a pesar de no haber obtenido el primer lugar en ninguna ocasión.

Problema de prueba	Punto de referencia
Binh	50.1 50.1
Deb1	1.1 1.1
Deb2	0.9 1.1
Deb3	1.1 1.1
Fonseca1	1.1 1.1
Fonseca2	1.1 1.1
Laumanns	4.1 4.1
Lis	0.92 0.84
Murata	4.1 4.1
ZDT1	1.1 1.1
ZDT2	1.1 1.1
ZDT3	0.9 1.1
ZDT4	1.1 1.1
ZDT6	1.1 1.1

Tabla 6.2: Puntos de referencia utilizados para el cálculo de hipervolumen

6.4. Comparación de desempeño

En esta sección se presentan los resultados obtenidos por los algoritmos y se discute el desempeño de nuestra propuesta. Los resultados son presentados en tablas comparativas, donde cada tabla muestra el desempeño de los algoritmos sobre el conjunto de pruebas para un indicador de desempeño en específico. Cada fila de la tabla representa a un problema de prueba distinto y las columnas representan al algoritmo en particular. En las celdas se muestra la media y la desviación estándar de las treinta ejecuciones independientes de cada resultado del indicador. Se utilizó una escala de grises para indicar a los mejores resultados de tal forma que el mejor resultado se encuentra resaltado con el tono más oscuro.. La ausencia de color, indica que el resultado no es estadísticamente significativo y por lo tanto, no es posible establecer una conclusión en ese caso.

Los puntos de referencia utilizados para calcular el hipervolumen se muestran en la tabla 6.2 y en el apéndice C se encuentran los frentes de Pareto que corresponden a la mediana del HV para cada problema y algoritmo comparado.

6.4.1. Comparación de MOPSOs mediante HV

En la tabla 6-1 se muestran los resultados obtenidos por los MOPSOs usados en nuestro estudio comparativo con respecto al indicador de hipervolumen. Al evaluar el hipervolumen, se debe considerar que es una métrica que nos indica la porción del espacio objetivo dominada por nuestras soluciones, por lo que entre más grande sea el valor obtenido, mejor es el desempeño de un algoritmo. Claramente, se destaca el rendimiento del SMPSO, que obtuvo el primero lugar en ocho de los problemas utilizados. En segundo lugar, se encuentra nuestra propuesta que obtuvo el primer lugar en cinco ocasiones. Cabe destacar que nuestra propuesta obtuvo cuatro de estos primeros lugares en los problemas de prueba ZDT1 a ZDT4, por lo que se puede decir que tuvo un buen desempeño en este conjunto de problemas. Cabe destacar que en cuatro ocasiones, no fue posible comparar las medias de nuestra propuesta con las de SMPSO, debido a que se determinó que no fue un resultado estadísticamente significativo mediante la prueba Wilcoxon. La tabla 6-2 muestra el orden de los algoritmos con respecto a HV utilizando el procedimiento descrito en la sección anterior. Hay que destacar que nuestra propuesta está compitiendo con dos algoritmos poblacionales, utilizando el mismo número de evaluaciones de función.

6.4.2. Comparación de AEMOs compactos mediante HV

En la tabla 6-3 se muestran los resultados obtenidos por los AEMOs compactos con respecto al indicador de hipervolumen. Como se puede observar, nuestra propuesta obtiene el primer lugar en nueve ocasiones, mientras que mocDE lo obtiene en cuatro ocasiones. Cabe destacar el desempeño de mocDE en problemas con un frente de Pareto desconectado (Deb2 y ZDT3), donde tuvo una victoria clara sobre nuestra propuesta. Respecto a compactMOPSO, obtuvo un desempeño superior en las funciones Binh, Laumanns y ZDT4. Que haya tenido un buen resultado para el problema ZDT4, nos indica que nuestra propuesta puede ser apta en problemas multi-frontales. Esto es algo que vale la pena destacar, considerando que nuestra propuesta es un AEMO sin población además de que mocDE no logró convergencia en este problema después de realizar 20,000 evaluaciones. En la tabla 6-4 se muestra el orden de los AEMOs conforme a su desempeño. Como se puede

HV	compactMOPSO	SMPSO	dMOPSO
binh	2080.345660 (0.086067)	2084.745125 (0.089551)	2080.262541 (0.081066)
deb1	0.538489 (0.000001)	0.538768 (0.000017)	0.538445 (0.000184)
deb2	0.984969 (0.000006)	0.985986 (0.000020)	0.984292 (0.000008)
deb3	0.280226 (0.000001)	0.280884 (0.000046)	0.280243 (0.000001)
fonseca1	0.272069 (0.000013)	0.272958 (0.000027)	0.272100 (0.000010)
fonseca2	0.547391 (0.000020)	0.547541 (0.000035)	0.547425 (0.000013)
laumanns	14.061445 (0.000203)	14.086216 (0.000943)	14.059843 (0.000217)
lis	0.523729 (0.002113)	0.483804 (0.005022)	0.527092 (0.001687)
murata	3.146369 (0.000045)	3.146362 (0.000069)	3.144908 (0.000024)
zdt1	0.871574 (0.000005)	0.871630 (0.000173)	0.871275 (0.000144)
zdt2	0.538489 (0.000001)	0.538454 (0.000174)	0.538294 (0.000047)
zdt3	1.327975 (0.000007)	1.326108 (0.068996)	1.324509 (0.000546)
zdt4	0.870843 (0.001259)	0.870945 (0.000393)	0.870069 (0.000851)
zdt6	0.504572 (0.000001)	0.515904 (0.139041)	0.504174 (0.000673)

Figura 6-1: Tabla comparativa que muestra los resultados obtenidos por los MOPSOs con respecto al indicador de hipervolumen, donde el mejor resultado se encuentra resaltado con el tono más oscuro.

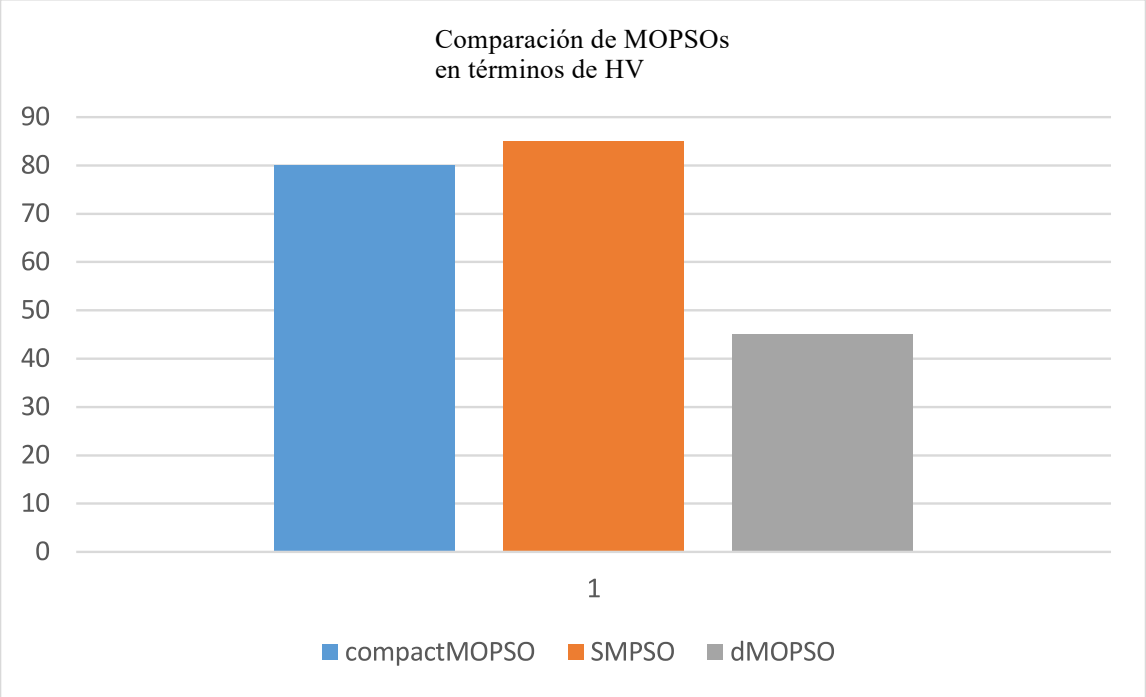


Figura 6-2: Clasificación de los MOPSOs con respecto a su desempeño medido mediante el indicador HV

HV	compactMOPSO	MOCDE
binh	2080.345660 (0.086067)	2070.566014 (1.545972)
deb1	0.538489 (0.000001)	0.538050 (0.000039)
deb2	0.984969 (0.000006)	1.035871 (0.160952)
deb3	0.280226 (0.000001)	0.282617 (0.014601)
fonseca1	0.272069 (0.000013)	0.278440 (0.007371)
fonseca2	0.547391 (0.000020)	0.541182 (0.003453)
laumanns	14.061445 (0.000203)	12.939475 (0.482121)
lis	0.523729 (0.002113)	0.419784 (0.059171)
murata	3.146369 (0.000045)	3.140816 (0.000251)
zdt1	0.871574 (0.000005)	0.867511 (0.001148)
zdt2	0.538489 (0.000001)	0.538207 (0.002395)
zdt3	1.327975 (0.000007)	2.005953 (0.050731)
zdt4	0.870843 (0.001259)	0.000000 (0.000000)
zdt6	0.504572 (0.000001)	0.464850 (0.017585)

Figura 6-3: Tabla comparativa que muestra los resultados obtenidos por AEMOs compactos con respecto al indicador de hipervolumen, donde el mejor resultado se encuentra resaltado con el tono más oscuro.

observar, nuestra propuesta queda en primer puesto con un porcentaje de obtención del primer lugar superior al 60 %.

6.4.3. Comparación de MOPSOs mediante SP

En la tabla 6-5 se muestran los resultados obtenidos por los MOPSOs con respecto al indicador de espaciado. El indicador de espaciado permite conocer el grado de dispersión de las soluciones, de tal forma que entre menor sea el valor, mejor distribuidas están las soluciones. Con el uso de este indicador, queda clara la ventaja que ofrece el uso de la distancia de agrupamiento para seleccionar los líderes que adopta SMPSO, el cual obtiene el primer lugar en trece de catorce problemas de prueba, siendo superado sólo en el problema ZDT6 por nuestra propuesta. A pesar de lo anterior y considerando que tanto compactMOPSO como dMOPSO utilizan un enfoque por descomposición mediante Tchebycheff, nuestra propuesta se mantuvo estable en el segundo lugar, quedando únicamente en el tercer lugar

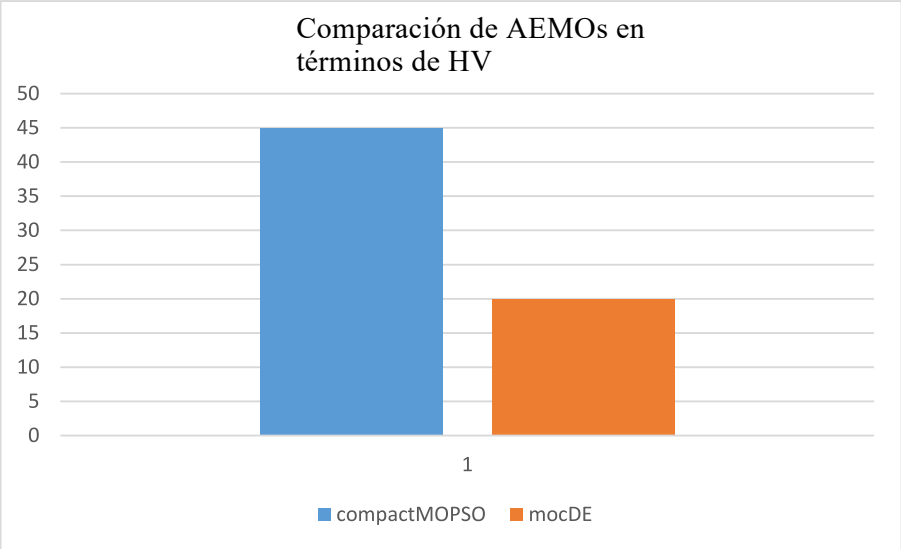


Figura 6-4: Clasificación de los AEMOs compactos con respecto a su desempeño medido mediante el indicador HV

en el problema ZDT4 por una pequeña diferencia. Los resultados de nuestra propuesta son muy competitivos respecto a los del SMPSO. Sin embargo, se puede observar que tanto compactMOPSO como dMOPSO tienen un mal desempeño para el problema Binh. En la tabla 6-6 se muestra el orden de los MOPSOs conforme a su desempeño, con respecto al espaciado, SMPSO domina por completo el primer lugar, mientras que nuestra propuesta obtuvo también un muy buen desempeño en comparación con dMOPSO.

6.4.4. Comparación de AEMOs compactos mediante SP

En la tabla 6-7 se muestran los resultados obtenidos por los AEMOs compactos con respecto al indicador de espaciado. En este caso, nuestra propuesta obtuvo el primer lugar en doce de catorce pruebas. En la comparación de los MOPSOs mediante SP, se discutió que tanto compactMOPSO como dMOPSO obtuvieron malos resultados para el problema binh. Dado que nuestra propuesta utiliza un enfoque de descomposición similar al de mocDE, y considerando que mocDE también tuvo malos resultados en este problema, el desempeño se puede atribuir al uso de un enfoque de descomposición. En la tabla 6-8 se muestra el orden de los AEMOs conforme a su desempeño. Para este caso, se vuelve a observar un mejor desempeño por parte de nuestro algoritmo, a pesar de que ambos algoritmos

SP	compactMOPSO	SMPSO	dMOPSO
binh	1.255077 (0.008260)	0.068229 (0.011203)	1.289514 (0.013618)
deb1	0.004213 (0.000010)	0.001194 (0.000205)	0.004782 (0.002989)
deb2	0.015639 (0.000092)	0.003179 (0.000351)	0.017584 (0.000058)
deb3	0.006879 (0.000016)	0.001706 (0.000689)	0.006890 (0.000011)
fonseca1	0.007712 (0.000034)	0.001337 (0.000293)	0.007833 (0.000073)
fonseca2	0.003950 (0.000041)	0.001329 (0.000195)	0.004002 (0.000033)
laumanns	0.100494 (0.000275)	0.007582 (0.001094)	0.103513 (0.000415)
lis	0.007025 (0.000721)	0.002778 (0.000945)	0.009053 (0.001273)
murata	0.005204 (0.000408)	0.003094 (0.000520)	0.011381 (0.000099)
zdt1	0.010045 (0.000040)	0.001634 (0.000266)	0.011060 (0.001288)
zdt2	0.004221 (0.000022)	0.001485 (0.000253)	0.004313 (0.000258)
zdt3	0.016925 (0.000057)	0.006011 (0.003096)	0.023636 (0.000868)
zdt4	0.010106 (0.000567)	0.002238 (0.000365)	0.009938 (0.000181)
zdt6	0.002057 (0.000020)	0.021369 (0.015597)	0.053270 (0.142027)

Figura 6-5: Tabla comparativa que muestra los resultados obtenidos por los MOPSOs con respecto al indicador de espaciado, donde el mejor resultado se encuentra resaltado con el tono más oscuro.

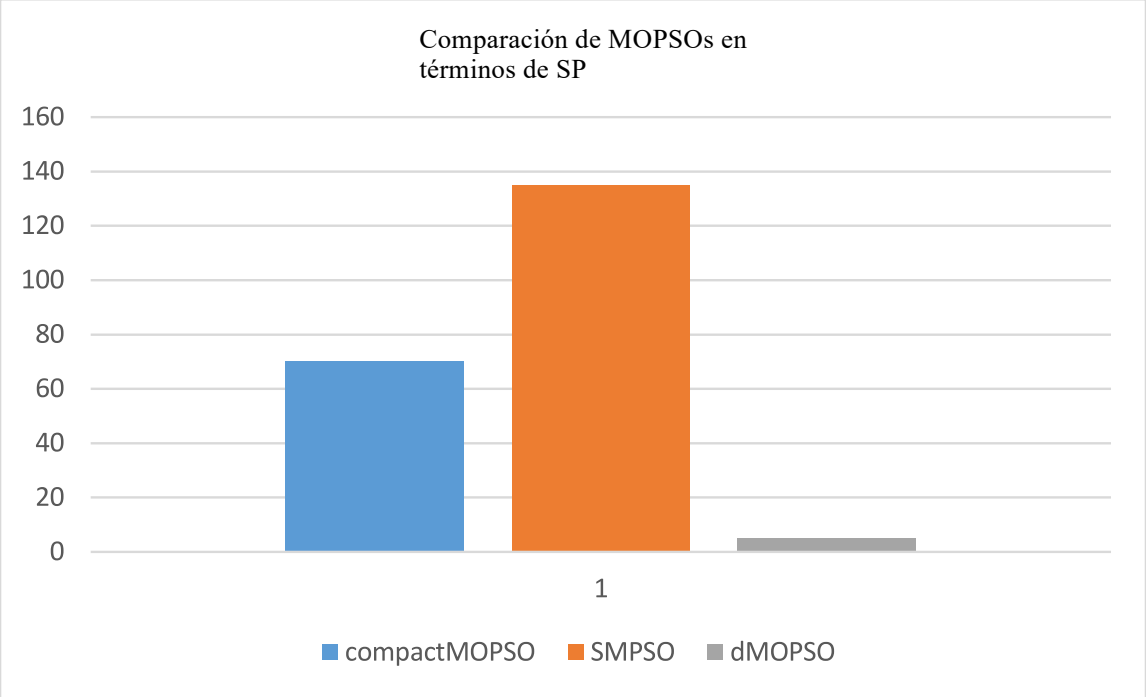


Figura 6-6: Clasificación de los MOPSOs con respecto a su desempeño medido mediante el indicador SP

SP	compactMOPSO	MOCDE
binh	1.255077 (0.008260)	1.552369 (0.119910)
deb1	0.004213 (0.000010)	0.005329 (0.000218)
deb2	0.015639 (0.000092)	0.025052 (0.017504)
deb3	0.006879 (0.000016)	0.008425 (0.003782)
fonseca1	0.007712 (0.000034)	0.012917 (0.002963)
fonseca2	0.003950 (0.000041)	0.010921 (0.001545)
laumanns	0.100494 (0.000275)	0.070851 (0.079104)
lis	0.007025 (0.000721)	0.007644 (0.009618)
murata	0.005204 (0.000408)	0.015474 (0.000545)
zdt1	0.010045 (0.000040)	0.014918 (0.001834)
zdt2	0.004221 (0.000022)	0.011008 (0.001633)
zdt3	0.016925 (0.000057)	0.025262 (0.001973)
zdt4	0.010106 (0.000567)	0.396595 (0.740801)
zdt6	0.002057 (0.000020)	0.060257 (0.101237)

Figura 6-7: Tabla comparativa que muestra los resultados obtenidos por los AEMOs compactos con respecto al indicador de espaciado, donde el mejor resultado se encuentra resaltado con el tono más oscuro.

utilizan un enfoque de descomposición.

6.4.5. Comparación de MOPSOs mediante IGD+

En la tabla 6-9 se muestran los resultados obtenidos por los MOPSOs con respecto al indicador de distancia generacional invertida más. El indicador IGD+ permite conocer qué tan cercano es un conjunto aproximado de óptimos de Pareto, de un conjunto de referencia. Al usar este indicador se observa una competencia más reñida. A pesar de que SMPSO obtiene el primer lugar ocho veces, queda en último lugar casi para todos los problemas ZDT. Es en estos problemas donde nuestra propuesta tiene la delantera, pues obtiene cinco veces el primer lugar y seis veces el segundo. Los resultados se mantienen similares a los obtenidos por el HV, en donde nuestra propuesta mantiene el liderazgo para los problemas Deb1 y ZDT1 a ZDT4. Por tercera ocasión vuelve a obtener el segundo lugar, superando en las tres pruebas a dMOPSO. Finalmente, cabe destacar lo pequeñas

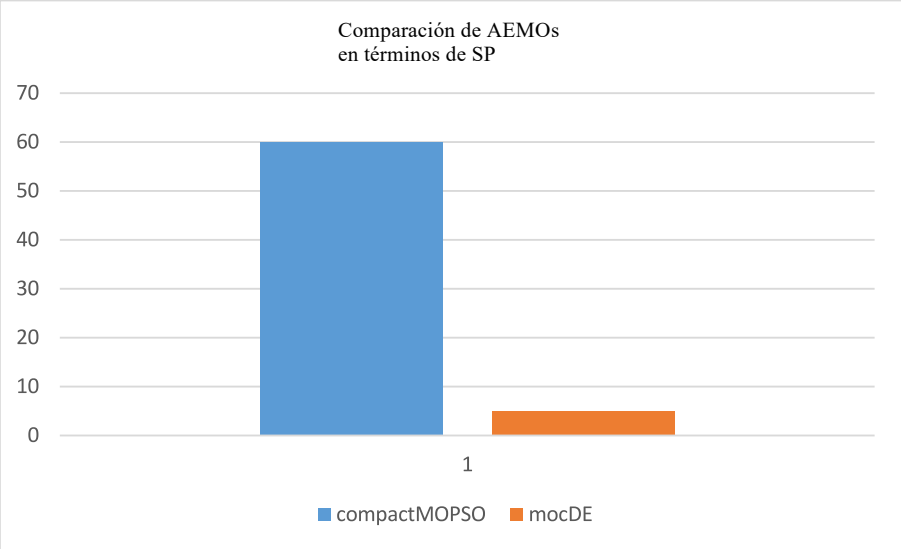


Figura 6-8: Clasificación de los AEMOs compactos con respecto a su desempeño medido con el indicador SP

que son las desviaciones estándar, lo que implica poca variabilidad en los resultados. En la tabla 6-10 se muestra el orden de los MOPSOs conforme a su desempeño, como ya se discutió. Al final, SMPSO obtuvo el primer lugar para los tres indicadores de desempeño,. Nuestra propuesta terminó con tres segundos lugares, pero, en indicadores como IGD+, se puede observar lo competitiva que es respecto a SMPSO.

6.4.6. Comparación de AEMOs compactos mediante IGD+

En la tabla 6-11 se muestran los resultados obtenidos por los AEMOs compactos con respecto al indicador IGD+. En este indicador, nuestra propuesta obtuvo el primer lugar en trece de catorce problemas de prueba, lo que muestra su superioridad respecto a mocDE. Como se discutió con respecto al indicador de hipervolumen, mocDE tiene una clara ventaja en funciones desconectadas respecto a nuestra propuesta y respecto a los MOPSOs antes mencionados. Esto probablemente se debe a la gran capacidad de exploración presente en la evolución diferencial. En la tabla 6-12 se muestra el orden de los AEMOs conforme a su desempeño. Por tercera ocasión nuestro algoritmo obtuvo el primer lugar, por lo cual, puede considerarse una alternativa útil a mocDE.

IGD+	compactMOPSO	SMPSO	dMOPSO
binh	0.132765 (0.001583)	0.082647 (0.015560)	0.133024 (0.001364)
deb1	0.002141 (0.000008)	0.002275 (0.000266)	0.002151 (0.000020)
deb2	0.001350 (0.000017)	0.000666 (0.000100)	0.001580 (0.000013)
deb3	0.001381 (0.000005)	0.001099 (0.000123)	0.001518 (0.000004)
fonseca1	0.001413 (0.000014)	0.000998 (0.000213)	0.001401 (0.000009)
fonseca2	0.002388 (0.000017)	0.002180 (0.000297)	0.002379 (0.000012)
laumanns	0.010461 (0.000033)	0.007024 (0.000789)	0.010488 (0.000053)
lis	0.002238 (0.000091)	0.001408 (0.000102)	0.002213 (0.000087)
murata	0.006220 (0.000086)	0.005706 (0.001028)	0.005985 (0.000046)
zdt1	0.002350 (0.000008)	0.002539 (0.000261)	0.002482 (0.000050)
zdt2	0.000057 (0.000007)	0.002344 (0.000181)	0.000241 (0.000112)
zdt3	0.001770 (0.000014)	0.005537 (0.015443)	0.003261 (0.000107)
zdt4	0.002681 (0.000663)	0.002827 (0.000333)	0.003049 (0.000435)
zdt6	0.001961 (0.000010)	0.010784 (0.035322)	0.001921 (0.000037)

Figura 6-9: Tabla comparativa que muestra los resultados obtenidos por los MOPSOs con respecto al indicador IGD+, donde el mejor resultado se encuentra resaltado con el tono más oscuro.

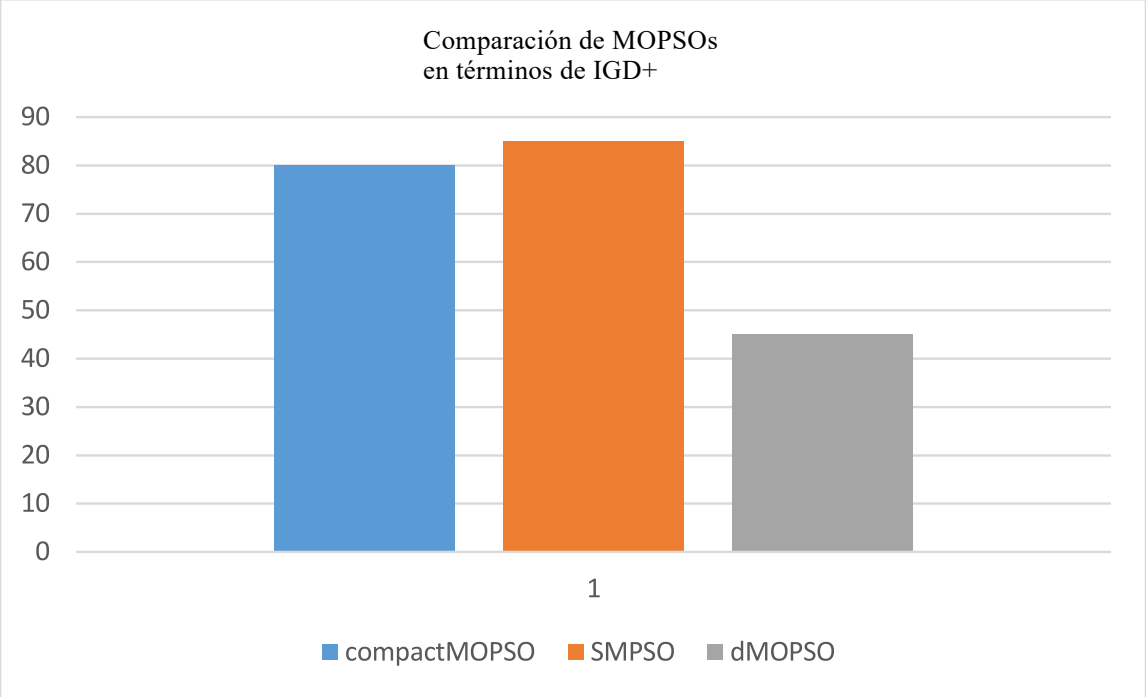


Figura 6-10: Clasificación de los MOPSOs con respecto a su desempeño medido con el indicador IGD+

IGD+	compactMOPSO	MOCDE
binh	0.132765 (0.001583)	0.233960 (0.014480)
deb1	0.002141 (0.000008)	0.002421 (0.000054)
deb2	0.001350 (0.000017)	0.002057 (0.000618)
deb3	0.001381 (0.000005)	0.001598 (0.000216)
fonseca1	0.001413 (0.000014)	0.002112 (0.000514)
fonseca2	0.002388 (0.000017)	0.006499 (0.001147)
laumanns	0.010461 (0.000033)	0.153165 (0.064276)
lis	0.002238 (0.000091)	0.021887 (0.008995)
murata	0.006220 (0.000086)	0.006702 (0.000113)
zdt1	0.002350 (0.000008)	0.005026 (0.000500)
zdt2	0.000057 (0.000007)	0.005149 (0.000737)
zdt3	0.001770 (0.000014)	0.001050 (0.000268)
zdt4	0.002681 (0.000663)	89.278937 (11.044041)
zdt6	0.001961 (0.000010)	0.026168 (0.011180)

Figura 6-11: Tabla comparativa que muestra los resultados obtenidos por los AEMOs compactos con respecto al indicador IGD+, donde el mejor resultado se encuentra resaltado con el tono más oscuro.

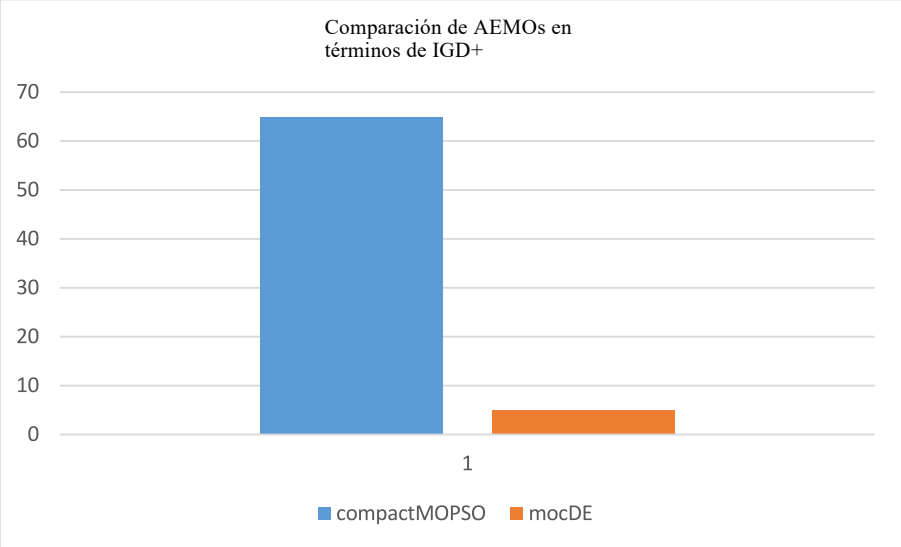


Figura 6-12: Clasificación de los AEMOs compactos con respecto a su desempeño medido con el indicador IGD+

6.5. Resumen

En este capítulo se presentó el estudio experimiental que se llevó a cabo para comparar el desempeño de nuestra propuesta respecto a tres metaheurísticas multi-objetivo del estado del arte (SMPSO, dMOPSO y mocDE). A partir de este estudio, se pudo concluir que nuestra propuesta presenta un rendimiento competitivo para la resolución de problemas de optimización con dos objetivos, obteniendo el segundo lugar en la categoría de MOPSOs siendo superado por SMPSO y superando a mocDE para los problemas de prueba adoptados en este trabajo de tesis.

Se observó que nuestra propuesta tiene un buen desempeño tanto en frentes con geometría cóncava como convexa al resolver satisfactoriamente los conjuntos de problemas ZDT y Deb. Sin embargo, se observó que en problemas que presentan un frente desconectado (Laumanns, ZDT3 y Deb2), nuestra propuesta tiene una mala distribución de las soluciones, ya que éstas se concentran sobre una región en específico sin importar si se normaliza al espacio de funciones objetivo. Esto también se pudo observar en dMOPSO y mocDE, por lo que se puede intuir que el enfoque de descomposición no obtiene un buen rendimiento en esta clase de problemas. Nuestra propuesta obtuvo muy buenos resultados en los problemas ZDT4 y ZDT6, lo que sugiere que puede ser apta en problemas multifrontales. Esto es algo que vale la pena destacar, considerando que nuestra propuesta no utiliza población.

Aunque el enfoque de descomposición es capaz de resolver problemas de optimización multi-objetivo con más de dos objetivos [55], nuestra propuesta, no fue capaz de resolver de forma satisfactoria el conjunto de problemas DTLZ [13] usando más de tres variables de decisión. A excepción del problema DTLZ6 (donde nuestra propuesta fue capaz de converger usando once variables de decisión en veinte mil evaluaciones), la diversidad y convergencia de las soluciones fue disminuyendo conforme se aumentaba el número de variables de decisión, siendo imposible para nuestra propuesta llenar las superficies que representan al frente de Pareto para dichos problemas. Esta es la razón por la que se decidió utilizar problemas únicamente de dos objetivos.

Cabe mencionar que existen dos problemas de mayor dificultad que fueron descartados en

este estudio experimental. Específicamente, se trata de los problemas de prueba Kursawe y Quagliarella ambos definidos en [7]. Estos problemas representan un gran reto para nuestra propuesta, ya que no fue capaz de obtener convergencia al frente óptimo de Pareto en ninguno de ellos, aún incrementando el número de evaluaciones de función. En el caso del problema Kursawe, todas las soluciones convergieron a un solo punto que no pertenece al frente óptimo de Pareto y en el caso de Quagliarella, nuestra propuesta quedó atrapada en un frente falso, mismo del que no se pudo escapar aún incrementando tanto la probabilidad de mutación como el número de evaluaciones de función.

Capítulo

7

Conclusiones y trabajo futuro

En este trabajo se propuso un PSO compacto para problemas de optimización multi-objetivo al que denominamos *compactMOPSO*. Se utilizó un enfoque de descomposición mediante el cual un problema multi-objetivo se transforma en un conjunto de subproblemas de métricas ponderadas. Para resolver estos subproblemas, se propuso un nuevo PSO compacto cuyo diseño está inspirado en el algoritmo genético compacto con elitismo persistente.

Al utilizar una representación estadística de la población, se eliminó la necesidad de almacenar al cúmulo de partículas, por lo que a diferencia de otros AEMOs (MOEA/D, SMPSO, MOPSO/D, etc.) que utilizan dos poblaciones: la población propia del algoritmo y el archivo externo para almacenar las soluciones no dominadas. Nuestra propuesta sólo requiere de un archivo externo para almacenar las soluciones no dominadas encontradas durante el proceso de búsqueda.

Los resultados experimentales muestran que compactMOPSO es capaz de resolver problemas complejos con dimensionalidad moderada considerando que es capaz de resolver de forma satisfactoria el conjunto de problemas de prueba ZDT con treinta variables de decisión, obteniendo los mejores resultados en el conjunto de problemas ZDT. A pesar de que SMPSO obtuvo los mejores resultados del estudio experimental, se observó que compactMOPSO es muy competitivo respecto a éste, y se observó que en varias ocasiones resultó superior a mocDE y dMOPSO para el conjunto de problemas de prueba utilizados.

Sin embargo, existen varios aspectos que se pueden mejorar. Uno de los aspectos más importantes que pudo comprometer el desempeño del algoritmo propuesto en ciertos problemas de prueba, es el uso de elitismo persistente. Al utilizar elitismo persistente, se tiene una fuerza de atracción tan grande que puede impedir la exploración de otras regiones del espacio de búsqueda incluso utilizando mutación y por ende, puede terminar sacrificando la diversidad de las soluciones. Esta es la razón principal por la que no se adopotó un esquema más complejo de mutación. Una forma de solucionar este problema, es incorporando el concepto de elitismo no persitente, que consiste en eliminar al lider después de un número de iteraciones sin éxito y reemplazarlo por una solución aleatoria. Sin embargo, esto puede traer una nueva serie de problemas que también deben ser estudiados.

En los resultados de su estudio experimental, Neri et al. [39] concluyeron que la exploración de un espacio dimensional alto es especialmente difícil para las metaheurísticas compactas, debido a que todas las soluciones son obtenidas mediante un mismo vector de probabilidades. Por ende, es posible obtener mejores resultados recurriendo a técnicas más complejas como lo pueden ser los algoritmos de estimación de distribución (EDAs, por sus siglas en inglés).

Apéndice

A

Problemas de prueba

A lo largo de los años se han propuesto diversos conjuntos de problemas de prueba para evaluar el desempeño de los algoritmos evolutivos multi-objetivo. Estos conjuntos están conformados por problemas que poseen una serie de características particulares que pueden dificultar el proceso de búsqueda de un algoritmo evolutivo multi-objetivo (AEMO) [56]. De tal forma, el uso de estos conjuntos de problemas de prueba permite analizar las posibles limitantes y/o ventajas presentadas por un algoritmo evolutivo multi-objetivo en particular, con respecto a otros.

Para medir el desempeño de nuestra propuesta se utilizaron catorce funciones de prueba que se describen a continuación.

A.1. Conjunto de problemas de prueba Zitzler-Deb-Thiele

El conjunto de problemas de prueba Zitzler-Deb-Thiele (ZDT) comprende seis problemas (ZDT1-ZDT6) de distintas características que pretenden validar diferentes capacidades de un AEMO.

Cada uno de estos problemas fue construido tomando en consideración la estructura propuesta por Deb en [14]. Para construir cada problema se requiere de tres funciones f_1, g y h las cuales pueden ser establecidas para crear distintos niveles de complejidad.

La función f_1 prueba la capacidad de un AEMO para encontrar diversas soluciones Pareto-

óptimas. La función g sirve para medir la capacidad de un AEMO para converger al frente óptimo de Pareto; esta función puede ser utilizada para crear POM's multi-frontales, deceptivos o aislados. La función h prueba la capacidad de un AEMO para resolver problemas donde el frente óptimo de Pareto es convexo, no-convexo o discreto.
Cada función mantiene la siguiente estructura.

$$\begin{aligned} &\text{Minimizar: } F(x) = (f_1, f_2) \\ &\text{Sujeto a: } f_2(x) = g(x_2, ..., x_n)h(f_1(x_1), g(x_2, ..., x_n)) \\ &\text{donde: } x = (x_1, ..., x_n) \end{aligned} \tag{A.1}$$

A continuación se define cada problema, a excepción del problema ZDT5 que requiere de codificación binaria y que, por lo tanto, no fue adoptado en los experimentos incluidos en esta tesis.

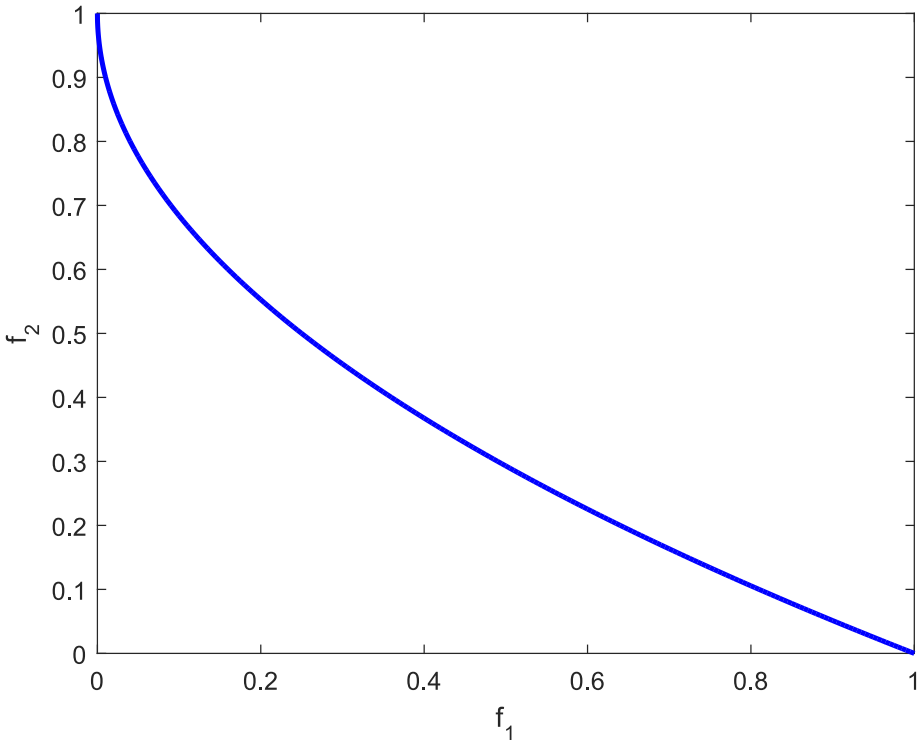


Figura A-1: Frente óptimo de Pareto para ZDT1

ZDT1

Este problema presenta un frente óptimo de Pareto convexo y se define de la siguiente forma:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, ..., x_n) &= 1 + 9 \cdot \sum_{i=2}^{n} \frac{x_i}{n-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} \end{aligned} \tag{A.2}$$

donde $n = 30$, $x_i \in [0, 1]$. El frente óptimo de Pareto se forma con $g(x) = 1$. En la figura A-1 se muestra el frente correspondiente.

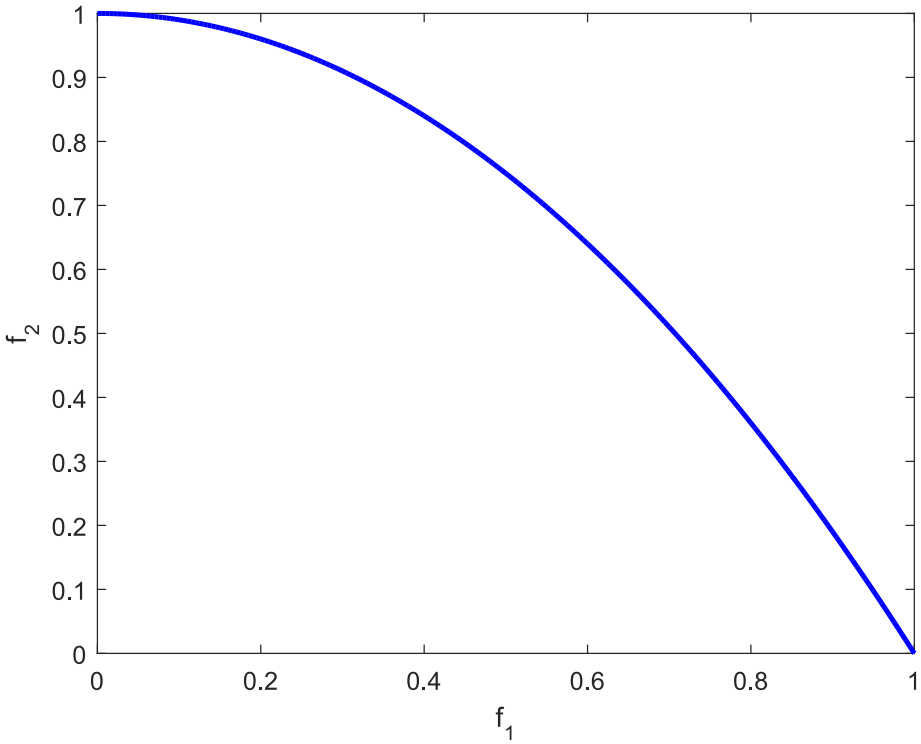


Figura A-2: Frente óptimo de Pareto para ZDT2

ZDT2

El problema de prueba ZDT2 es la contraparte no convexa de ZDT1:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, ..., x_n) &= 1 + 9 \cdot \sum_{i=2}^{n} \frac{x_i}{n-1} \\ h(f_1, g) &= 1 - \left(\frac{f_1}{g}\right)^2 \end{aligned} \tag{A.3}$$

donde $n = 30$, $x_i \in [0, 1]$. El frente óptimo de Pareto se forma con $g(x) = 1$. En la figura A-2 se muestra el frente correspondiente.

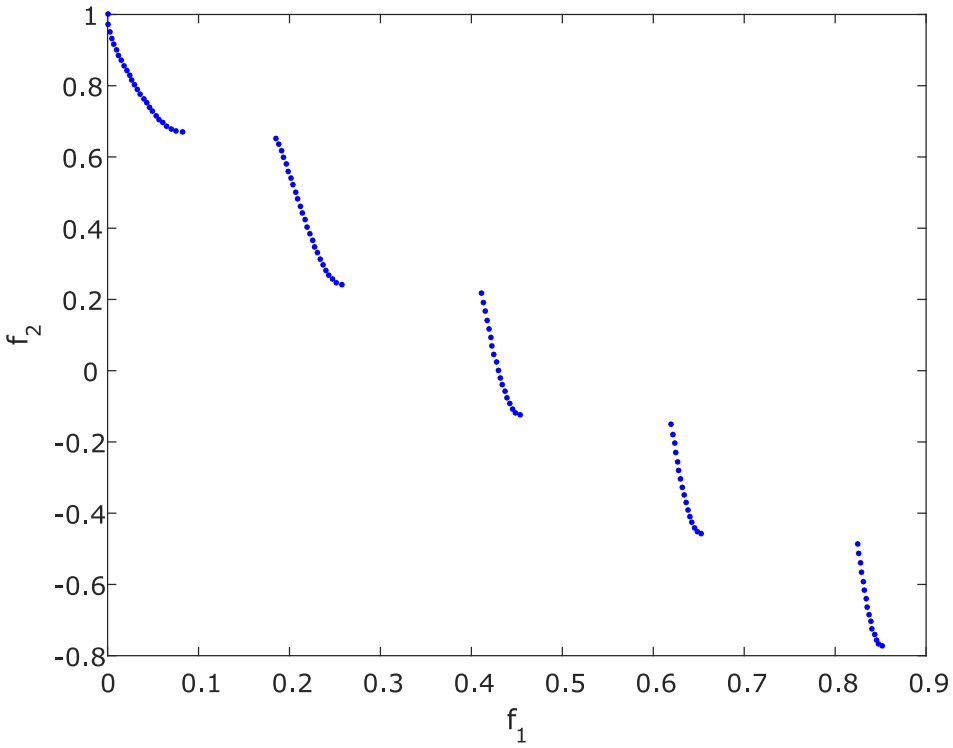


Figura A-3: Frente óptimo de Pareto para ZDT3

ZDT3

El problema de prueba ZDT3 presenta un frente óptimo de Pareto desconectado, que consiste en varias partes convexas no contiguas:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, ..., x_n) &= 1 + 9 \cdot \sum_{i=2}^{n} \frac{x_i}{n-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} - \left(\frac{f_1}{g}\right) sen(10\pi f_1) \end{aligned} \tag{A.4}$$

donde $n = 30$, $x_i \in [0, 1]$. El frente óptimo de Pareto se forma con $g(x) = 1$. La función seno presente en h causa discontinuidad en el frente óptimo de Pareto. Sin embargo, no existe discontinuidad en el espacio de las variables de decisión. En la figura A-3 se muestra el frente correspondiente.

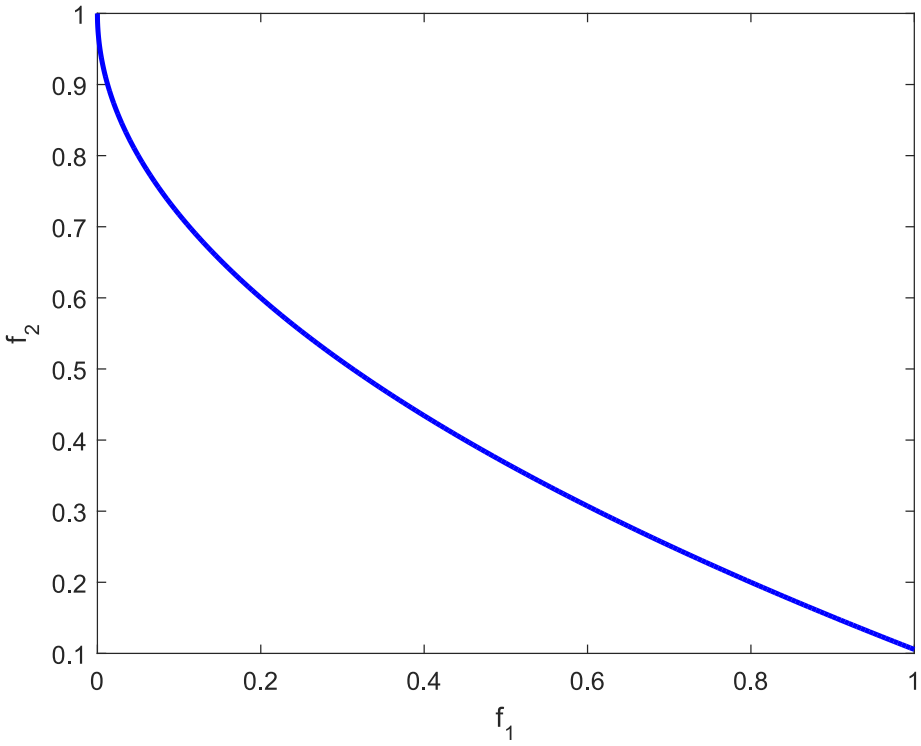


Figura A-4: Frente óptimo de Pareto para ZDT4

ZDT4

El problema de prueba ZDT4 contiene 21^9 frentes óptimos de Pareto locales. Por lo tanto este problema es muy útil para medir la capacidad de un AEMO para lidiar con problemas multi-frontales.

$$\begin{aligned} & f_1(x_1) = x_1 \\ & g(x_2, ..., x_n) = 1 + 10(n-1) + \sum_{i=2}^{n} x_i^2 - 10cos(4\pi x_i) \\ & h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} \end{aligned} \tag{A.5}$$

donde $n = 30$, $x_1 \in [0, 1]$, $x_2, ..., x_n \in [-5, 5]$. El mejor frente óptimo de Pareto local corresponde a $g(x) = 1.25$. En la figura A-4 se muestra el frente correspondiente.

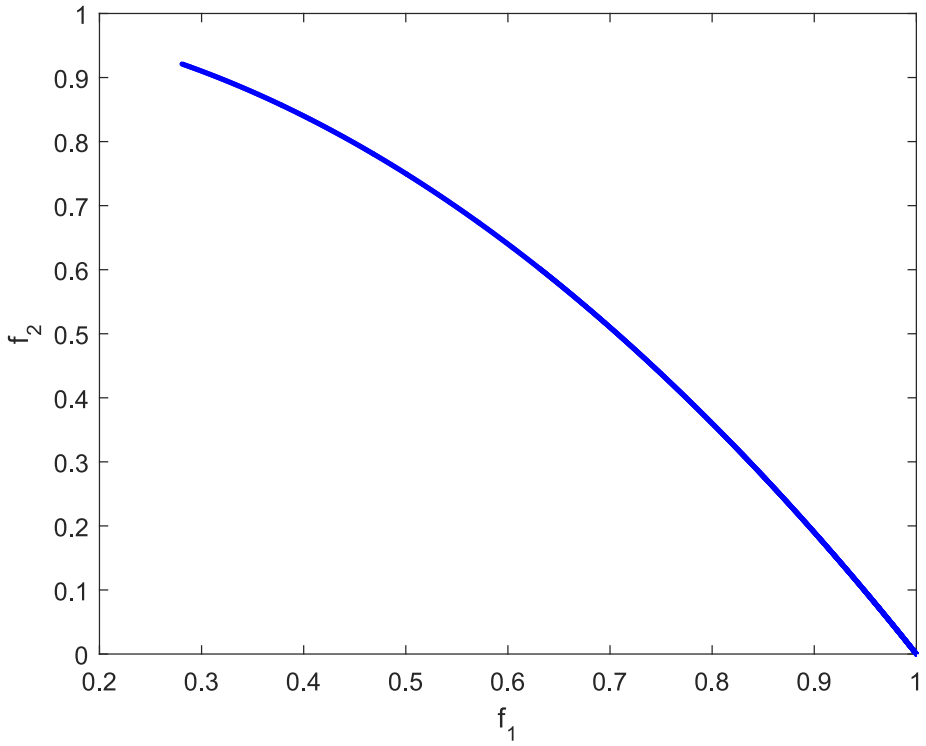


Figura A-5: Frente óptimo de Pareto para ZDT6

ZDT6

El problema de prueba ZDT6 presenta dos dificultades provocadas por la no uniformidad del espacio de búsqueda: la primera radica en que las soluciones óptimas de Pareto se encuentran distribuidas de forma no uniforme sobre el frente de Pareto global; y la segunda consiste en que la densidad de las soluciones disminuye conforme nos acercamos al frente de Pareto y aumenta conforme nos alejamos de éste.

$$
\begin{aligned}
f_1(x_1) &= 1 - exp(-4x_1)sen^6(6\pi x_1) \\
g(x_2, ..., x_n) &= 1 + \left(9 \cdot \sum_{i=2}^{n} \frac{x_i}{n-1}\right)^{0.25} \\
h(f_1, g) &= 1 - \left(\frac{f_1}{g}\right)^2
\end{aligned}
\tag{A.6}
$$

donde $n = 30$, $x_i \in [0, 1]$. El frente óptimo de Pareto se forma con $g(x) = 1$. En la figura A-5 se muestra el frente correspondiente.

A.2. Problemas de prueba estándar

A continuación se definen los problemas de prueba estándar restantes, utilizados para medir el desempeño de nuestra propuesta. La tabla A.1 obtenida de [7] muestra las distintas características presentes en los problemas de prueba que quedan por definir. Cada fila de la tabla corresponde a un problema de prueba a utilizar y cada columna indica una característica presente ya sea en el espacio de las variables de decisión o en el de las funciones objetivo. Los problemas multi-objetivo que exhiban cualquiera de esas características se encontrarán marcados con una x en la columna respectiva. Cuando un problema no tiene una marca x sobre cualquiera de las dos columnas que indican la característica 'Conectado' se asume que el espacio respectivo es desconectado. En caso de que el problema no tenga marcada la propiedad 'Convexo' se asume que el problema es cóncavo.

Espacio de las variables de decisión				Espacio de las funciones objetivo		
Problema	Conectado	Simétrica	# variables	Geometría	Conectado	Convexo
Bihn	x	x	$2\mathbb{R}$	Curva	x	x
Deb1	x	x	$2\mathbb{R}$	Curva	x	
Deb2		x	$2\mathbb{R}$	Curva		x
Deb3	x	x	$2\mathbb{R}$	Curva	x	
Fonseca	x	x	$2\mathbb{R}$	Curva	x	
Fonseca2	x	x	$2\mathbb{R}$	Curva	x	
Laumanns	x	x	$2\mathbb{R}$	Puntos		x
Lis	x	x	$2\mathbb{R}$	Curva	x	x
Murata	x	x	$2\mathbb{R}$	Curva	x	

Tabla A.1: Características de los problemas de prueba restantes.

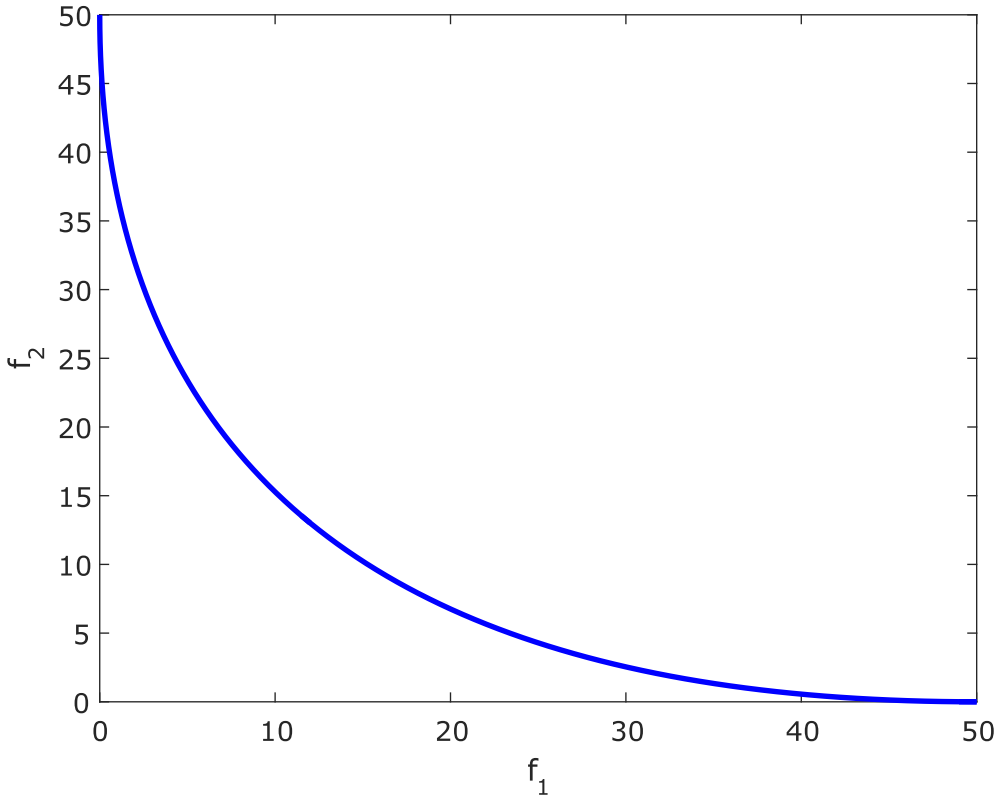


Figura A-6: Frente óptimo de Pareto para Binh

Binh

$$
\begin{aligned}
F &= (f_1(x,y), f_2(x,y)) \\
f_1(x,y) &= x^2 + y^2 \\
f_2(x,y) &= (x-5)^2 + (y-5)^2
\end{aligned}
\tag{A.7}
$$

donde $-5 \leq x, y \leq 10$. En la figura A-6 se muestra el frente correspondiente.

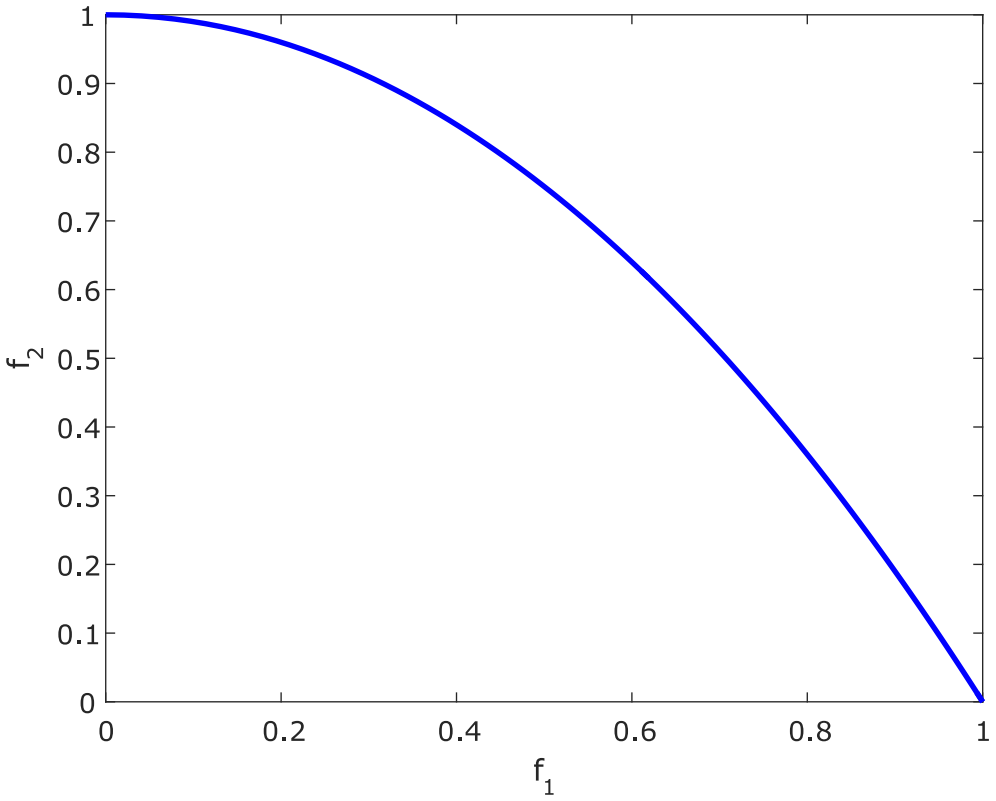


Figura A-7: Frente óptimo de Pareto para Deb1

Deb1

$$
\begin{aligned}
&F = (f_1(x), f_2(x)) \\
&f_1(x) = x_1 \\
&f_2(x, g) = g(x)h(x) \\
&g(x) = 1 + x_2^2 \\
&h(x) = \begin{cases} 1 - \left(\frac{f_1(x)}{g(x)}\right)^2 & \text{si } f_1 \leq g; \\ 0 & \text{de lo contrario} \end{cases}
\end{aligned}
\tag{A.8}
$$

donde $0 \leq x_i \leq 1, i = 1, 2$. En la figura A-7 se muestra el frente correspondiente.

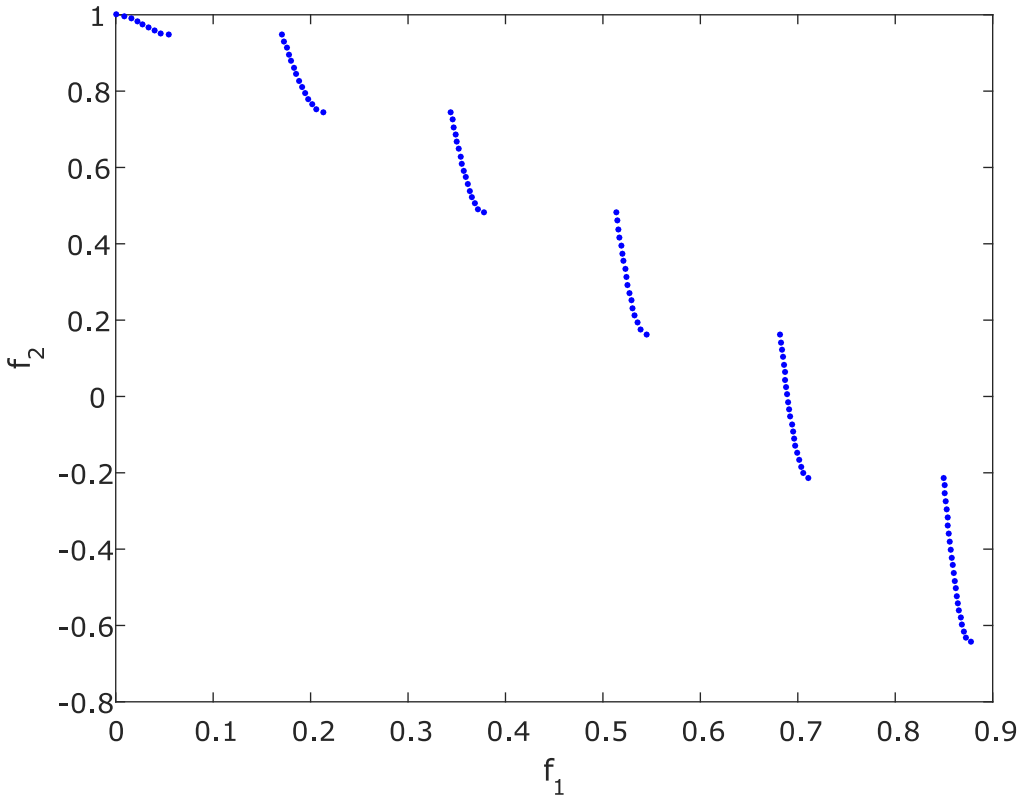


Figura A-8: Frente óptimo de Pareto para Deb2

Deb2

$$
\begin{aligned}
&F = (f_1(x), f_2(x)) \\
&f_1(x) = x_1 \\
&f_2(x, g) = g(x)h(x) \\
&g(x) = 1 + 10x_2 \\
&h(x) = 1 - \left(\frac{f_1}{g(x)}\right)^2 - \frac{f_1}{g(x)} \cdot sen(12\pi f_1)
\end{aligned}
\tag{A.9}
$$

donde $0 \leq x_i \leq 1, i = 1, 2$. En la figura A-8 se muestra el frente correspondiente.

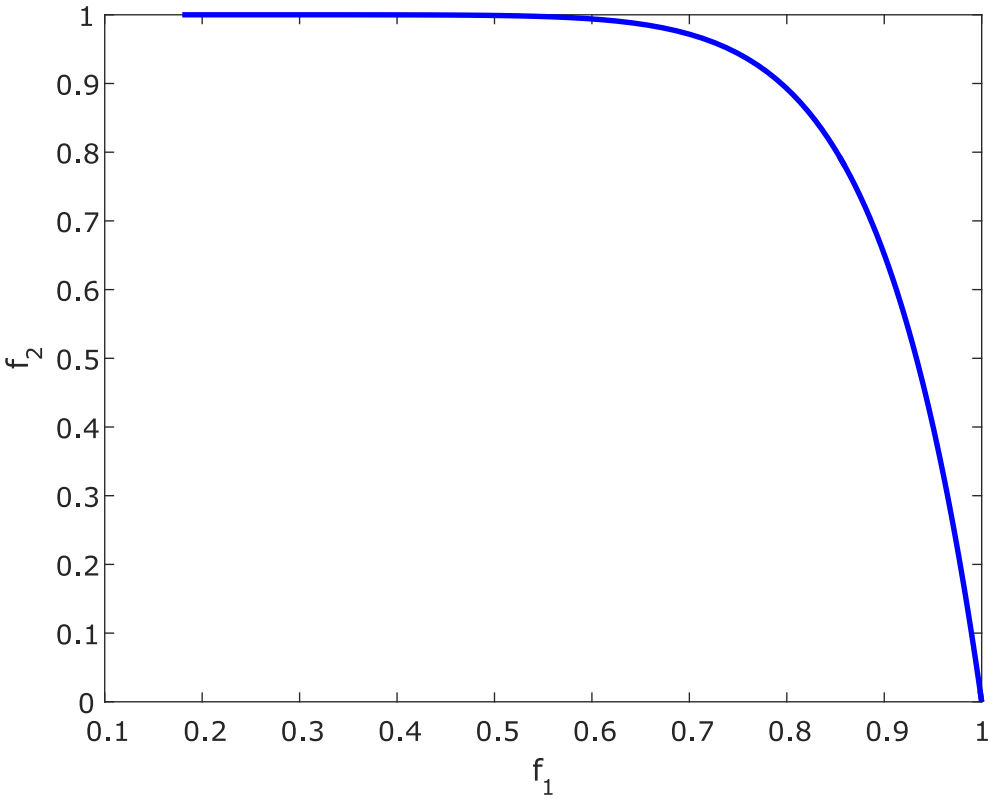


Figura A-9: Frente óptimo de Pareto para Deb3

Deb3

$$
\begin{aligned}
&F = (f_1(x), f_2(x)) \\
&f_1(x) = 1 - e^{-4x_1} sen^4(10\pi x_1) \\
&f_2(x, g) = g(x)h(x) \\
&g(x) = 1 + x_2^2 \\
&h(x) = \begin{cases} 1 - \left(\frac{f_1(x)}{g(x)}\right)^{10} & \text{si } f_1 \leq g; \\ 0 & \text{de lo contrario} \end{cases}
\end{aligned}
\tag{A.10}
$$

donde $0 \leq x_i \leq 1, i = 1, 2$. En la figura A-9 se muestra el frente correspondiente.

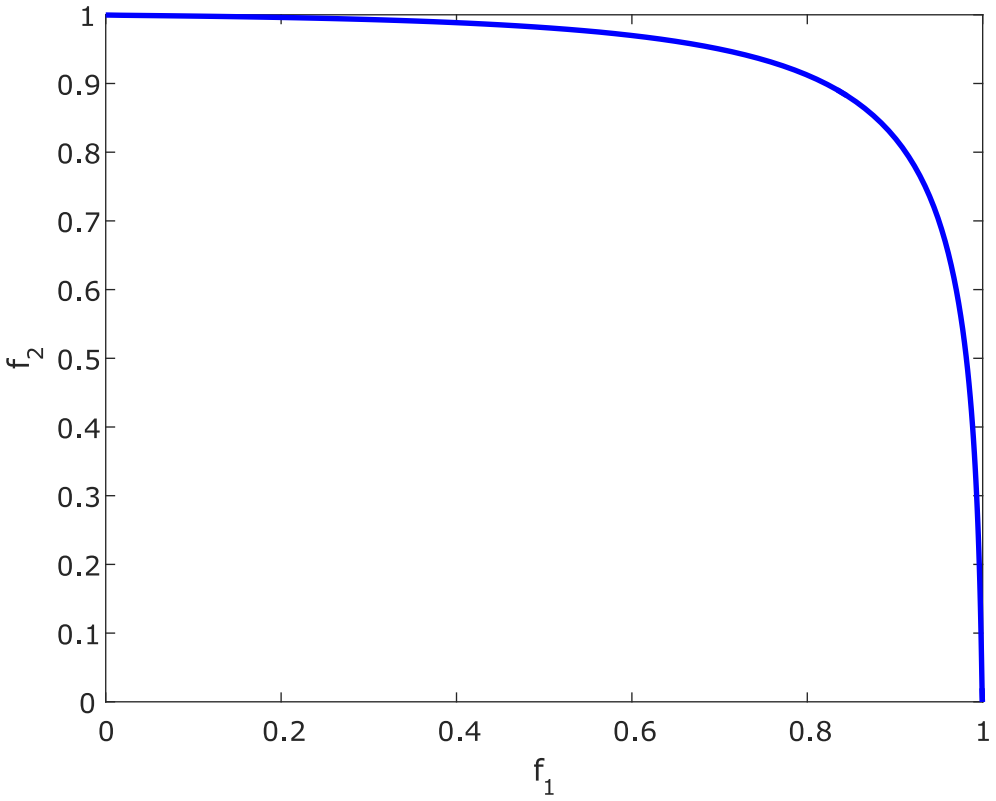


Figura A-10: Frente óptimo de Pareto para Fonseca1

Fonseca1

$$
\begin{aligned}
F &= (f_1(x,y), f_2(x,y)) \\
f_1(x,y) &= 1 - exp(-(x-1)^2 - (y+1)^2) \\
f_2(x,y) &= 1 - exp(-(x+1)^2 - (y-1)^2)
\end{aligned}
\tag{A.11}
$$

donde $-4 \leq x, y \leq 4$. En la figura A-10 se muestra el frente correspondiente.

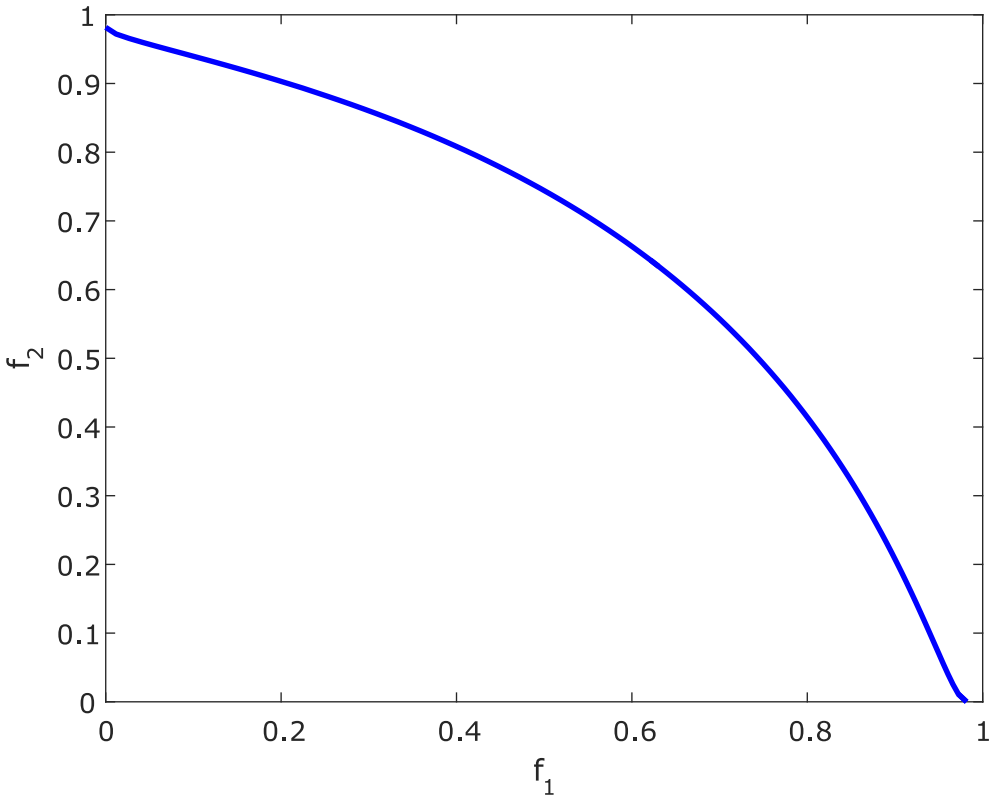


Figura A-11: Frente óptimo de Pareto para Fonseca2

Fonseca2

$$
\begin{aligned}
&F = (f_1(x), f_2(x)) \\
&f_1(x) = 1 - exp\left(-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2\right) \\
&f_2(x) = 1 - exp\left(-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2\right)
\end{aligned}
\tag{A.12}
$$

donde $n = 2$ y $-4 \leq x_1, x_2 \leq 4$. En la figura A-11 se muestra el frente correspondiente.

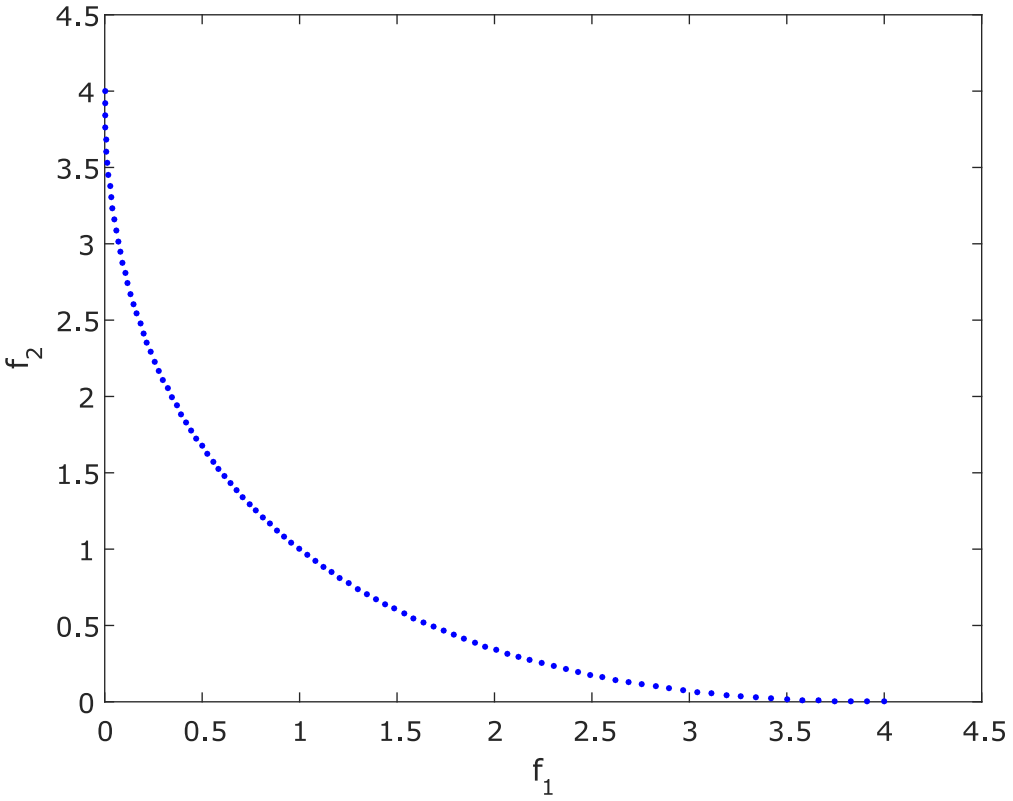


Figura A-12: Frente óptimo de Pareto para Laumanns

Laumanns

$$
\begin{aligned}
F &= (f_1(x,y), f_2(x,y)) \\
f_1(x,y) &= x^2 + y^2 \\
f_2(x,y) &= (x+2)^2 + y^2
\end{aligned}
\tag{A.13}
$$

donde $-50 \leq x, y \leq 50$. En la figura A-12 se muestra el frente correspondiente.

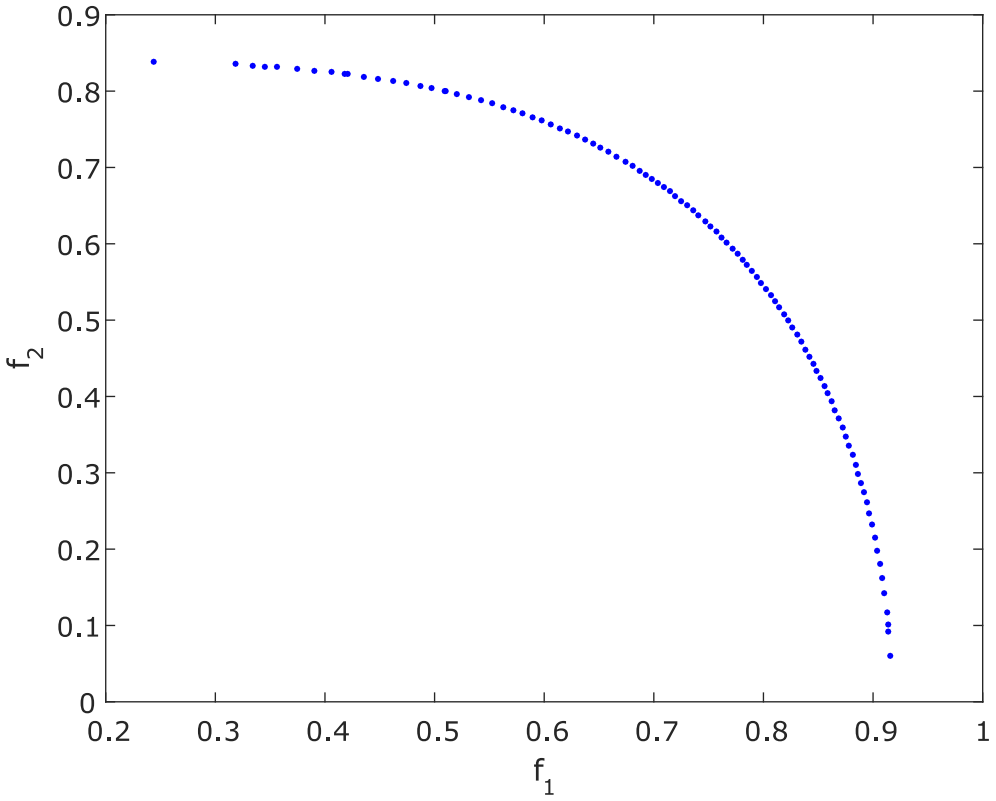


Figura A-13: Frente óptimo de Pareto para Lis

Lis

$$
\begin{aligned}
F &= (f_1(x,y), f_2(x,y)) \\
f_1(x,y) &= \sqrt[8]{x^2+y^2} \\
f_2(x,y) &= \sqrt[4]{(x-0.5)^2+(y-0.5)^2}
\end{aligned}
\tag{A.14}
$$

donde $-5 \leq x, y \leq 10$. En la figura A-13 se muestra el frente correspondiente.

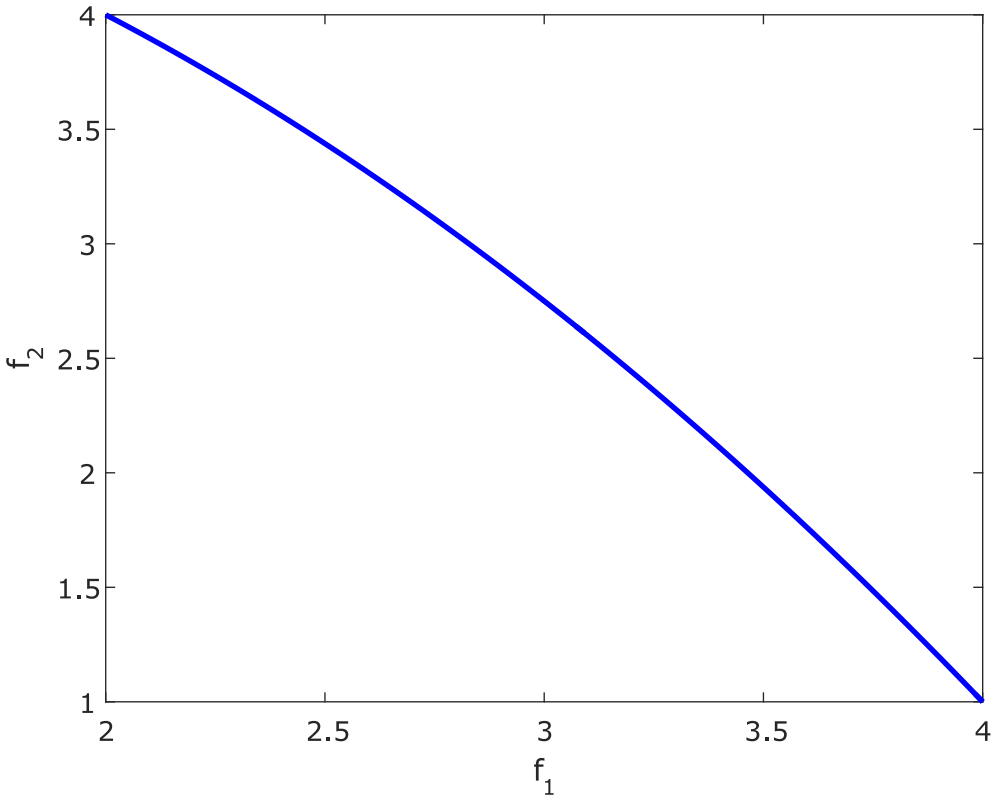


Figura A-14: Frente óptimo de Pareto para Murata

Murata

$$
\begin{aligned}
F &= (f_1(x,y), f_2(x,y)) \\
f_1(x,y) &= 2\sqrt{x} \\
f_2(x,y) &= x(1-y) + 5
\end{aligned}
\tag{A.15}
$$

donde $1 \leq x \leq 4$, $1 \leq y \leq 2$. En la figura A-14 se muestra el frente correspondiente.

Apéndice

B

Indicadores de desempeño para optimización multi-objetivo

El uso de indicadores de desempeño nos permite reducir la dimensión de un conjunto aproximado de óptimos de Pareto a una sola figura de mérito [57]. Una de las principales ventajas de utilizar indicadores, es que esta reducción a una dimensión permite que las pruebas estadísticas puedan llevarse a cabo de forma directa utilizando pruebas estadísticas de una forma similar a si se compararan las aptitudes de distintos algoritmos para problemas mono-objetivo.
A continuación se define el concepto de indicador de calidad unario.

Definición B.1. ***Indicador de calidad binario.*** *Es una función $I : \Psi \rightarrow \mathbb{R}$ que asigna a cada conjunto aproximado de Pareto un número real, Ψ es el espacio de las variables de decisión de un POM.*

En combinación con las relaciones $\leq$ o $\geq$ sobre $\mathbb{R}$, un indicador de calidad I define un orden total sobre el espacio objetivo Ω. Dados dos conjuntos de soluciones que aproximen óptimos de Pareto producidos por diferentes algoritmos A, B, la diferencia entre $I(A)$ y $I(B)$ debería revelar una diferencia en la calidad de los dos conjuntos. Esto también funciona cuando A y B son incomparables. Por lo tanto, esta información va más allá del concepto de la dominancia de Pareto, y representa información adicional, denotada como *información de preferencia* [57]. Se dice que A es preferible a B, si $I(A) > I(B)$, asumiendo que los valores del indicador son maximizados.

B.1. Hipervolumen

El indicador de hipervolumen fue introducido en [58] y proporciona el volumen de la porción del espacio objetivo que es débilmente dominada por una aproximación del conjunto de óptimos de Pareto. Formalmente se define como:

Definición B.2. ***Indicador de hipervolumen.***

$$I_H^*(A) := \int_{z_{inferior}}^{z_{superior}} \alpha_A(z)dz \tag{B.1}$$

donde $z_{inferior}$ y $z_{inferior}$ son los límites inferiores y superiores del espacio objetivo (vectores objetivo) en el que será calculado el hipervolumen, y donde la función α_A se define de la manera siguiente:

$$\alpha_A(z) = \begin{cases} 1 & \textit{si: } \exists x \in A : f(x) \preceq z \\ 0 & \textit{de lo contrario} \end{cases} \tag{B.2}$$

Generalmente no se requiere del límite inferior $z_{inferior}$ para calcular el hipervolumen de un conjunto A. El indicador de hipervolumen debe ser maximizado. Éste es el único indicador unario conocido que es estrictamente monótono de acuerdo a la dominancia de Pareto. Para calcular el hipervolumen, se utilizó la implementación propuesta en [21].

B.2. Espaciado

El indicador de Espaciado (S) propuesto por Jason Schott en 1995 [45], describe numéricamente la propagación de los vectores que constituyen la aproximación al frente de Pareto (PF_{known}). Específicamente, es la desviación estándar de las distancias mínimas para cada uno de los vectores de PF_{known} respecto a sus vecinos.

El primer problema consiste en encontrar la distancia mínima entre dos vectores i, j denotada mediante d_i.

$$d_i = \min_{j=1, i \neq j}^{n} \left(\sum_{k=1}^{m} |f_i^k - f_j^k| \right) \tag{B.3}$$

donde $n = |PF_{known}|$ y m es la dimensión del espacio objetivo. A partir de lo anterior, se

obtiene el promedio $\bar{d}$ y la desviación estándar mediante:

$$\begin{aligned} \bar{d} &= \frac{1}{n}\sum_{i=1}^{n} d_i \\ S &= \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{d} - d_i)^2} \end{aligned} \tag{B.4}$$

El valor de indicador debe ser minimizado. Cuando $S = 0$, implica que la media de todos los individuos de PF_{known} es idéntica, lo que significa que todos los individuos están igualmente separados uno del otro.

B.3. IGD+

El indicador denominado distancia generacional invertida más (IGD+), fue propuesto por Ishibuchi et al. [27]. Surge con el objetivo de solucionar un problema presente en el indicador IGD [7], que en general, consiste en encontrar la distancia promedio entre cada cada uno de los puntos de referencia (Z) con la solución del frente aproximado (A) más cercana a éstos. Pero a diferencia de IGD, el indicador IGD+ toma en consideración el concepto de dominancia de Pareto, al suponer que todas las soluciones del punto de referencia dominan al menos de forma débil a todas las soluciones del conjunto aproximado. Por lo tanto, al obtener la distancia euclidiana entre una solución perteneciente al conjunto de referencia y una solución del conjunto aproximado sobre una dimensión, se asume que la diferencia entre dimensiones siempre debe de ser positiva. Por lo tanto, se realiza una modificación al cálculo de la distancia euclidiana, de tal forma que se descarten todos los objetivos en los que la solución aproximada sea mejor que la solución de referencia.
A continuación se define la distancia entre un punto de referencia z y una solución del conjunto aproximado:

$$d_{IGD+}(a, z) = \sqrt{\sum_{k=1}^{m}(\text{máx}\{a_k - z_k, 0\})^2} \tag{B.5}$$

Ahora se presenta la definción formal del IGD+:

$$IGD+(A) = \frac{1}{|Z|}\sum_{j=1}^{|Z|} \min_{a \in A} d_{IGD+}(a, z_j) \tag{B.6}$$

Al igual que el indicador de espaciado, esta métrica debe de ser minimizada. Entre más cerca de cero se encuentre el valor obtenido después de aplicar el indicador, más cerca se encuentra el conjunto aproximado del conjunto de referencia.

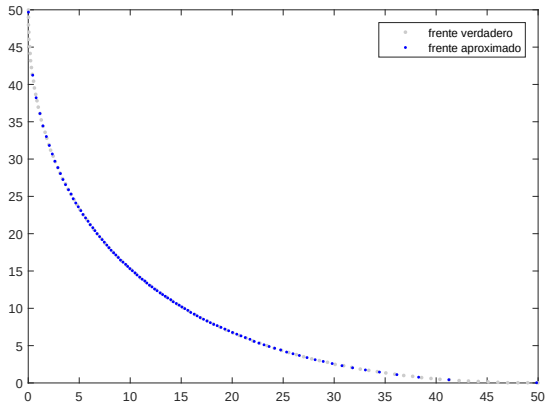
Apéndice

C

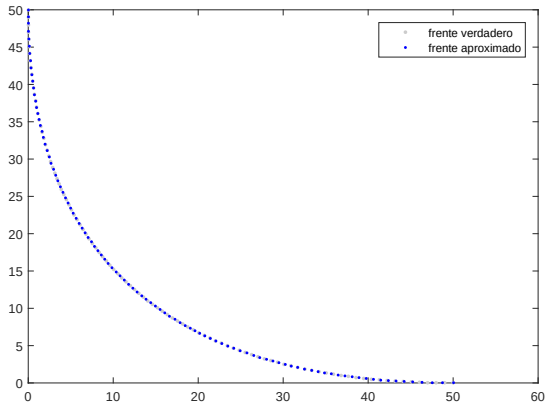
Resultados completos

Este apéndice tiene como propósito mostrar los frentes de Pareto de los algoritmos utilizados en el capítulo 6 para cada problema de prueba. Cada gráfica muestra el frente de Pareto correspondiente con la mediana de las treinta ejecuciones realizadas para el indicador de hipervolumen. También se presentan las gráficas de caja en las que se puede apreciar de forma gráfica el desempeño de cada uno de los algoritmos por cada indicador de desempeño utilizado. Las gráficas muestran las dos categorías empleadas para nuestro estudio expermiental: la comparación con respecto a algoritmos multi-objetivo basados en cúmulos de partículas y la comparación con respecto a AEMOs compactos.

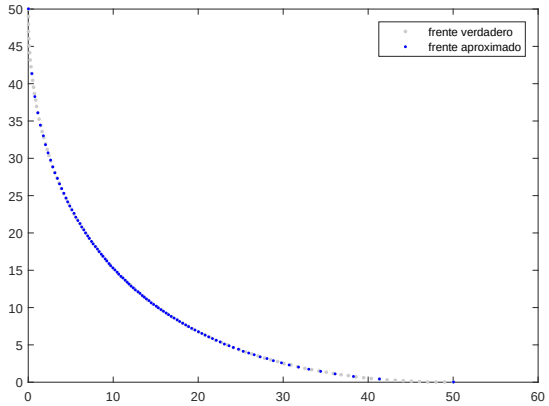
C.1. Gráficas frentes de Pareto



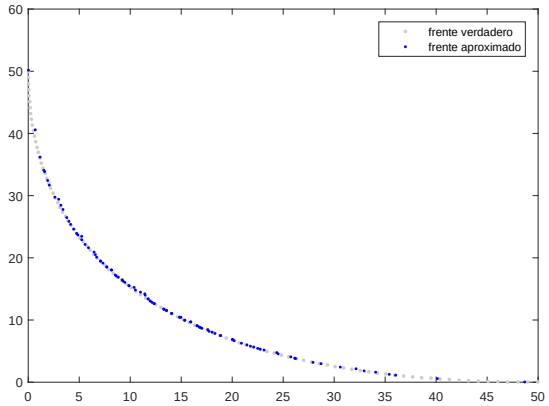
(a) compactMOPSO



(b) SMPSO

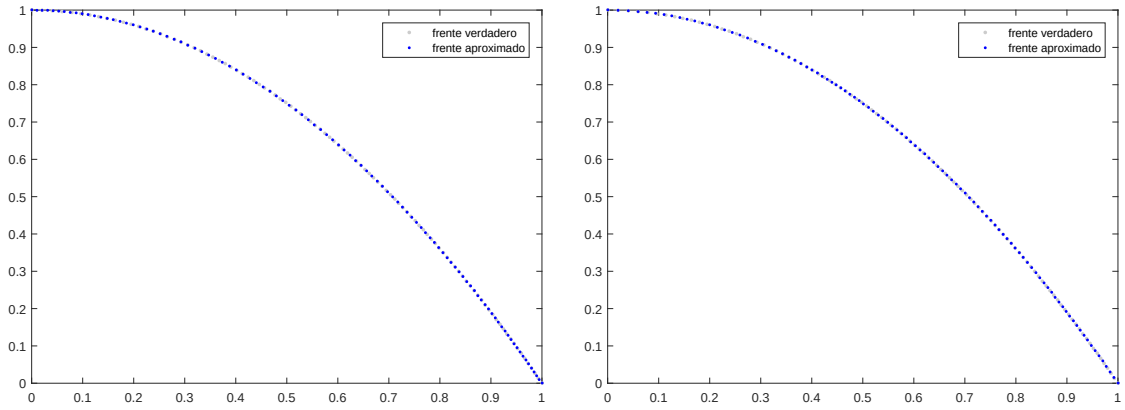


(c) dMOPSO



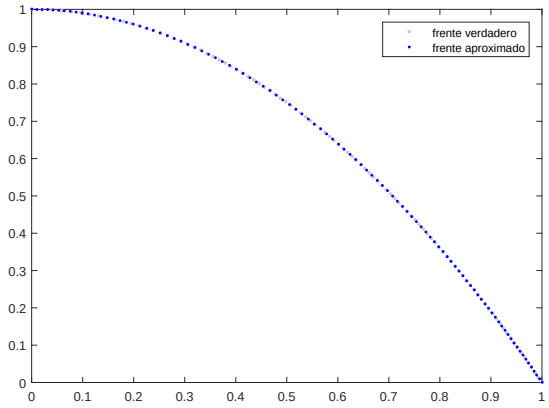
(d) mocDE

Figura C-1: Frentes de Pareto para el problema de prueba Binh

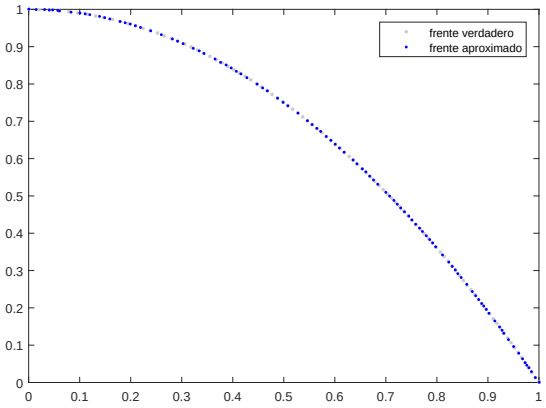


(a) compactMOPSO

(b) SMPSO

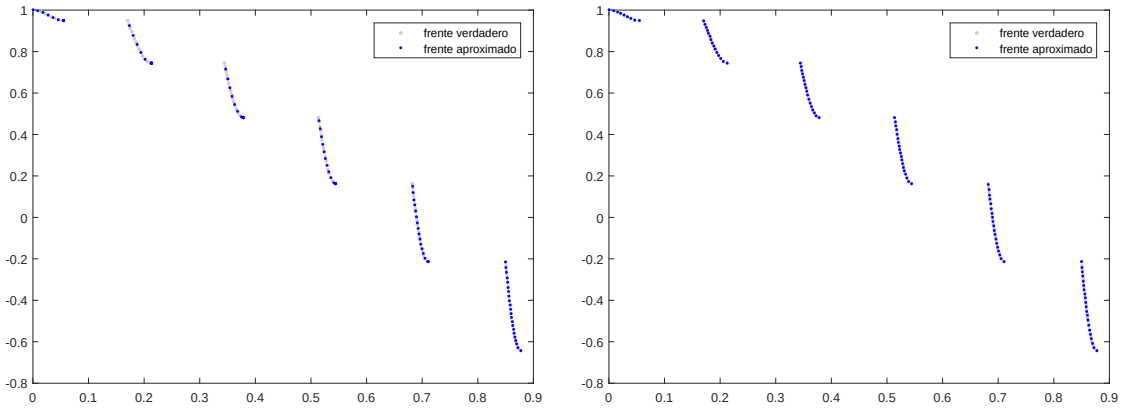


(c) dMOPSO



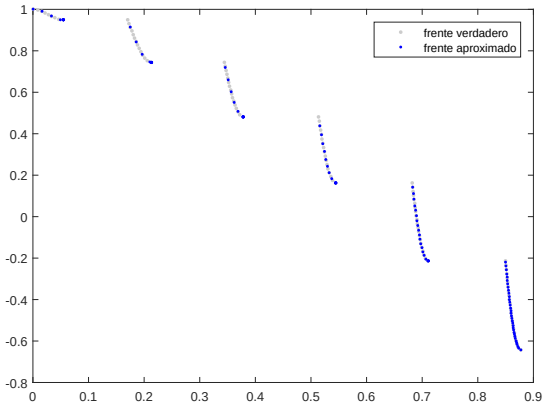
(d) mocDE

Figura C-2: Frentes de Pareto para el problema de prueba Deb1

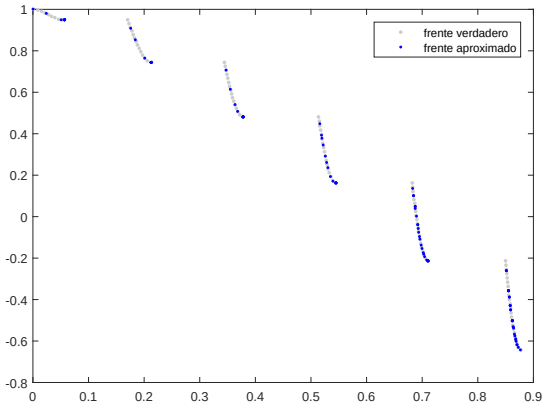


(a) compactMOPSO

(b) SMPSO

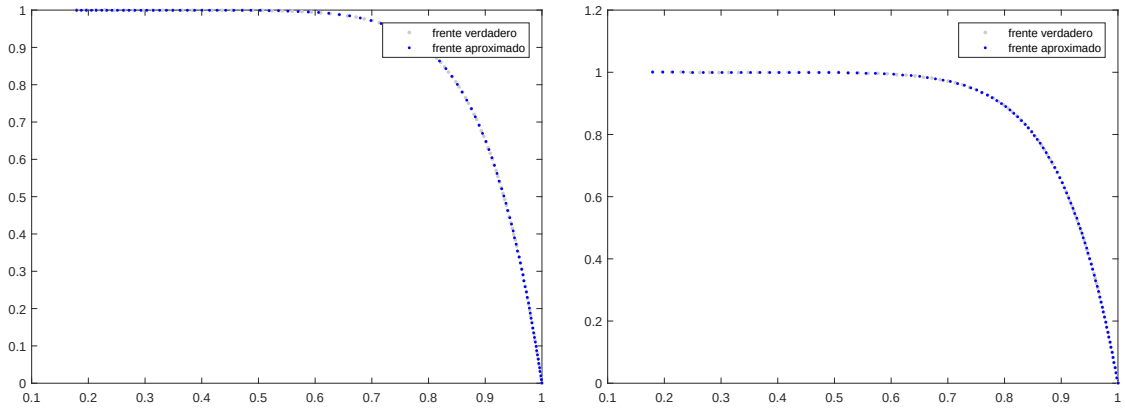


(c) dMOPSO



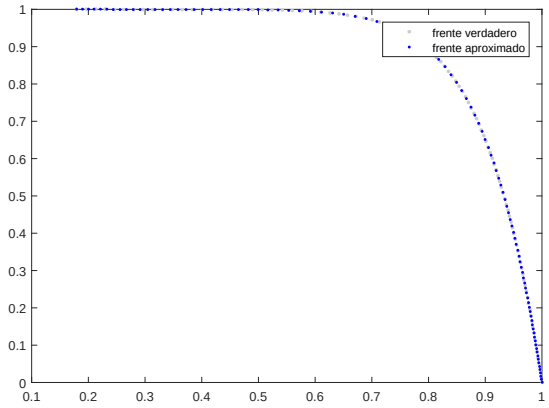
(d) mocDE

Figura C-3: Frentes de Pareto para el problema de prueba Deb2

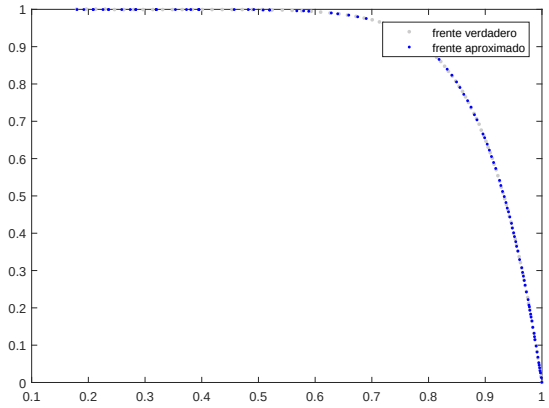


(a) compactMOPSO

(b) SMPSO

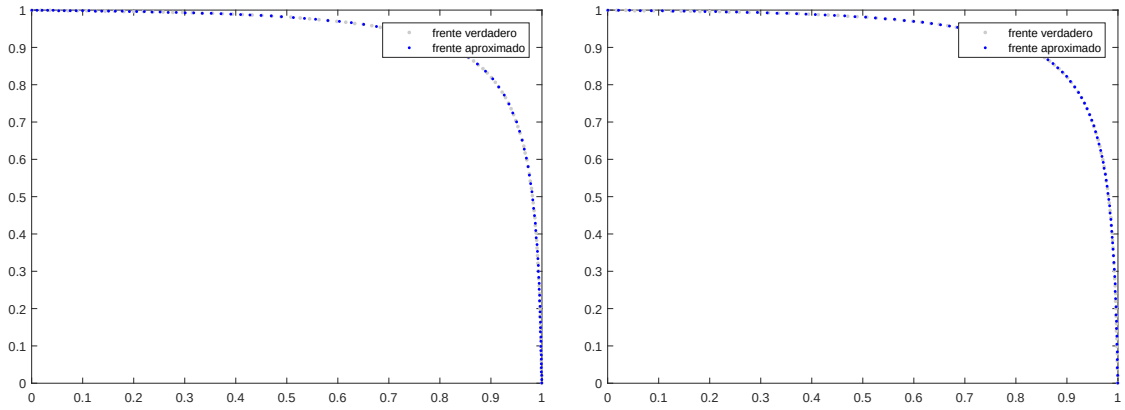


(c) dMOPSO



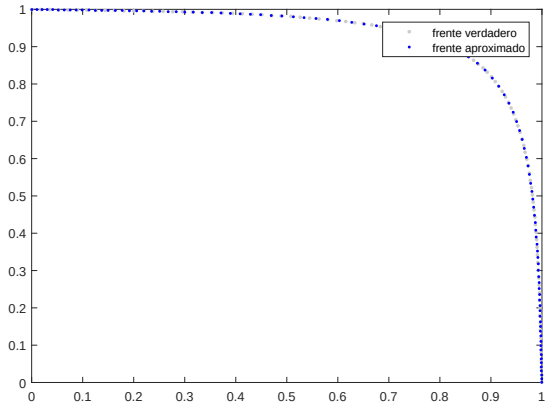
(d) mocDE

Figura C-4: Frentes de Pareto para el problema de prueba Deb3

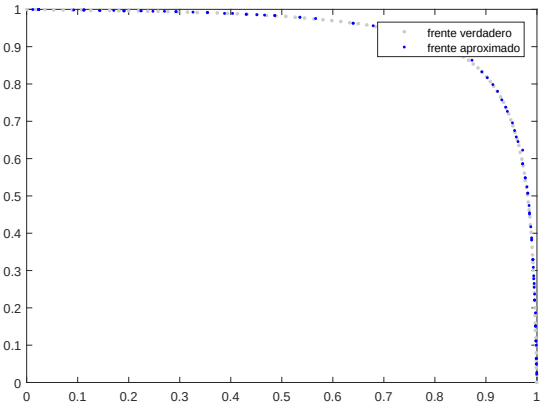


(a) compactMOPSO

(b) SMPSO

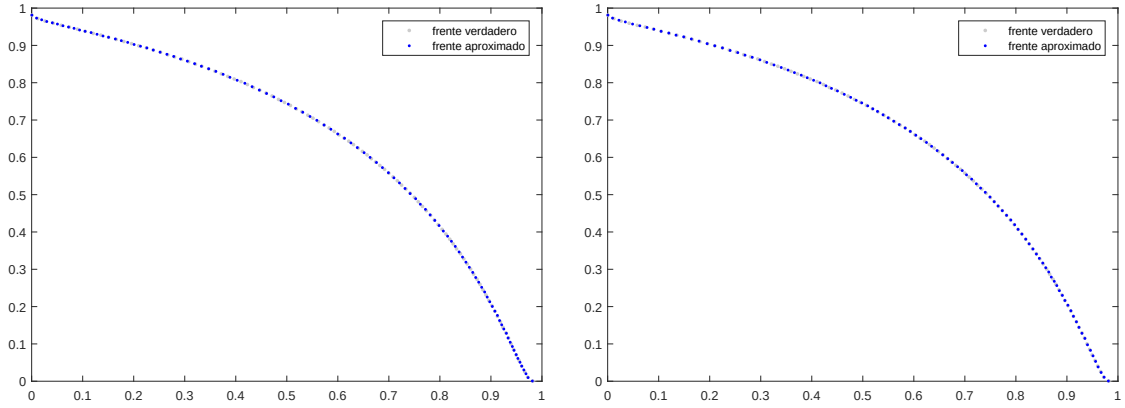


(c) dMOPSO



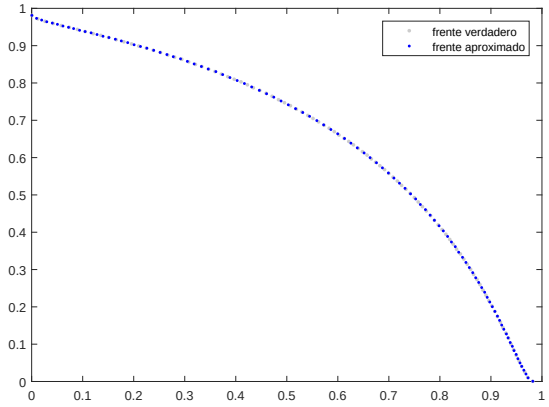
(d) mocDE

Figura C-5: Frentes de Pareto para el problema de prueba Fonseca1

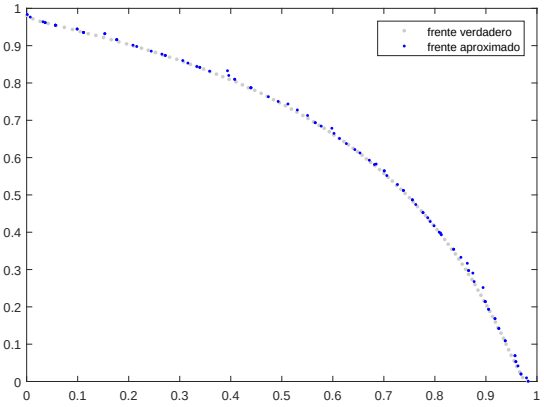


(a) compactMOPSO

(b) SMPSO

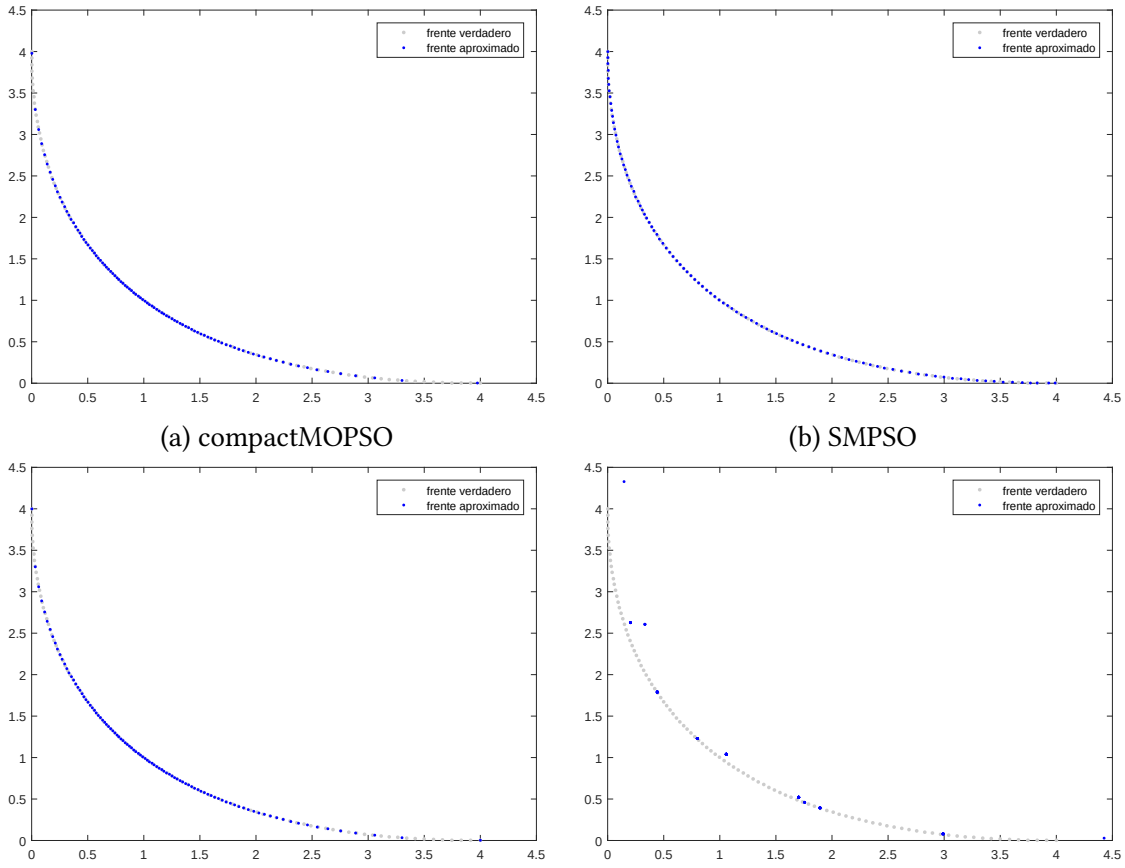


(c) dMOPSO



(d) mocDE

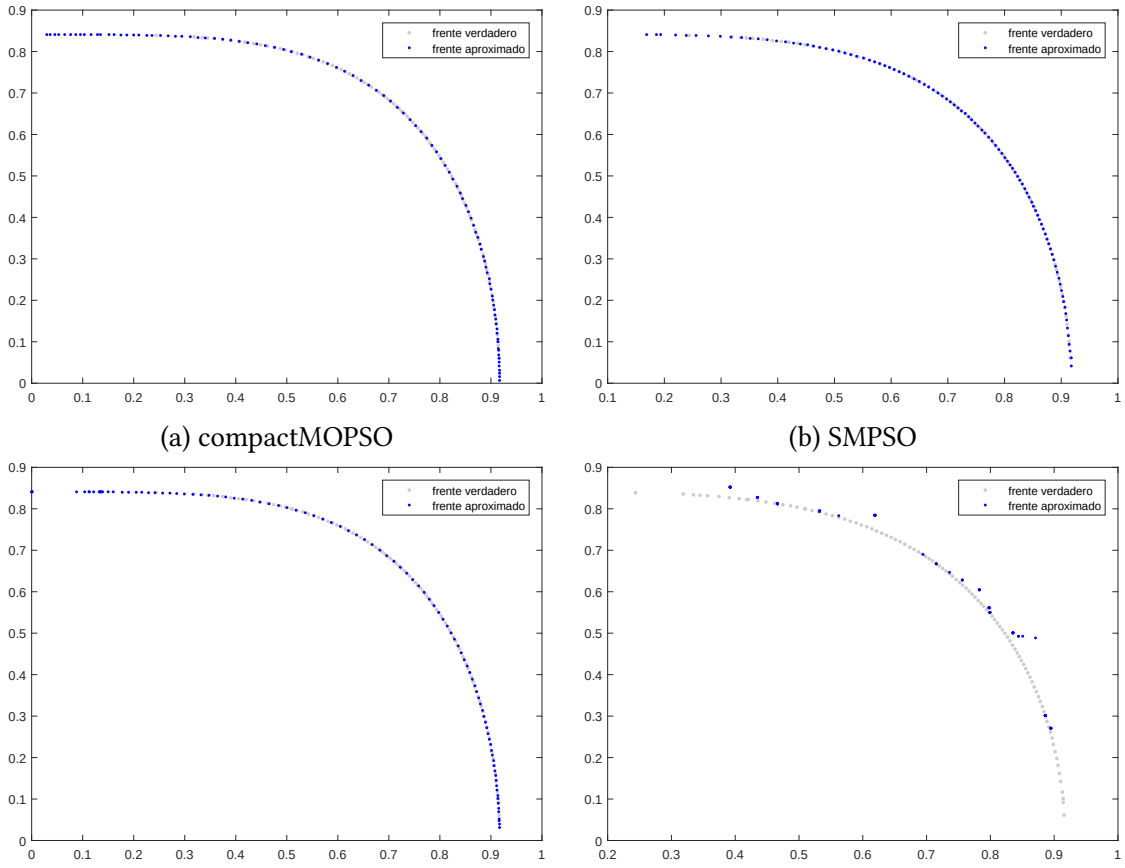
Figura C-6: Frentes de Pareto para el problema de prueba Fonseca2



(a) compactMOPSO (b) SMPSO

(c) dMOPSO (d) mocDE

Figura C-7: Frentes de Pareto para el problema de prueba Laumanns



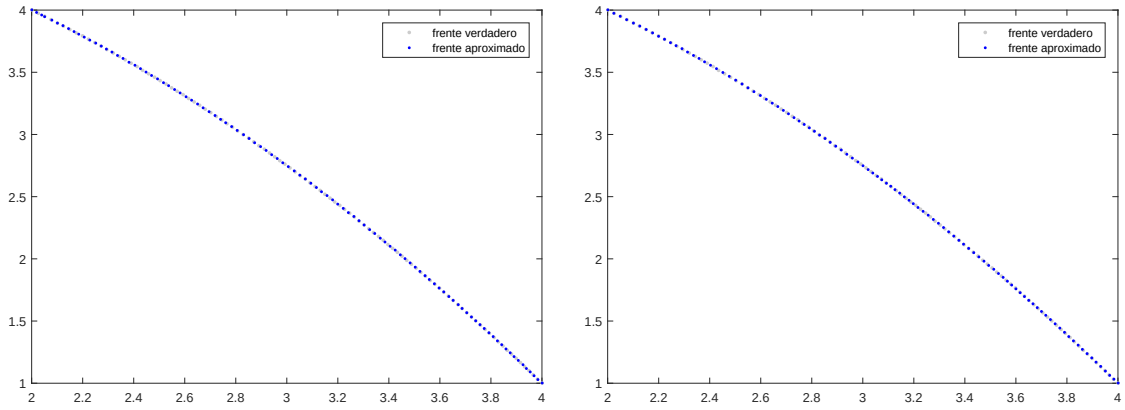
(a) compactMOPSO

(b) SMPSO

(c) dMOPSO

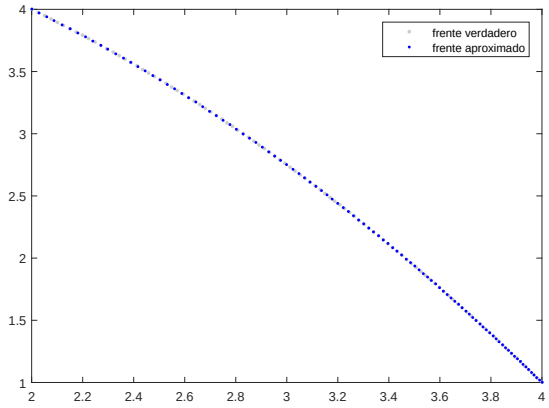
(d) mocDE

Figura C-8: Frentes de Pareto para el problema de prueba Lis

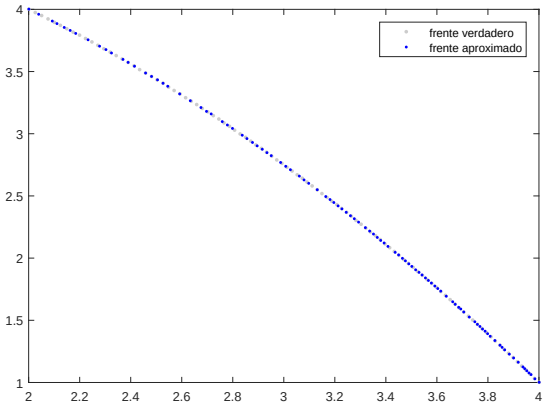


(a) compactMOPSO

(b) SMPSO

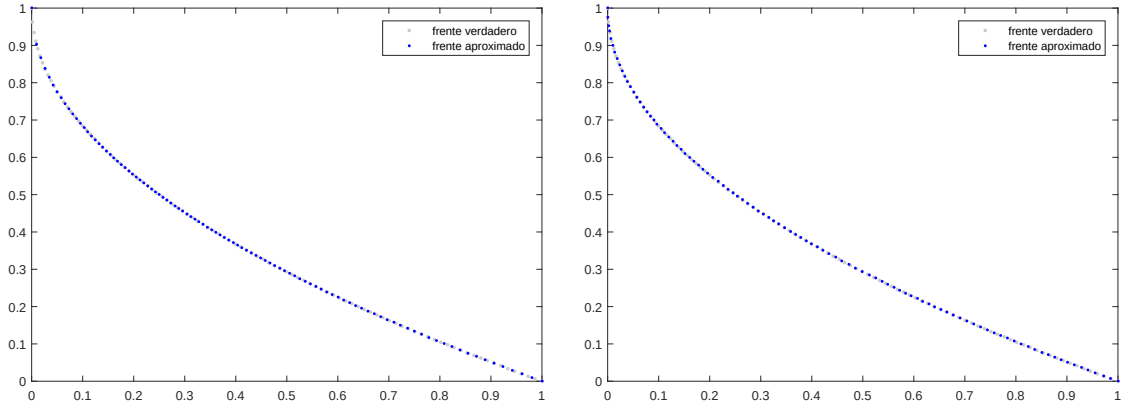


(c) dMOPSO



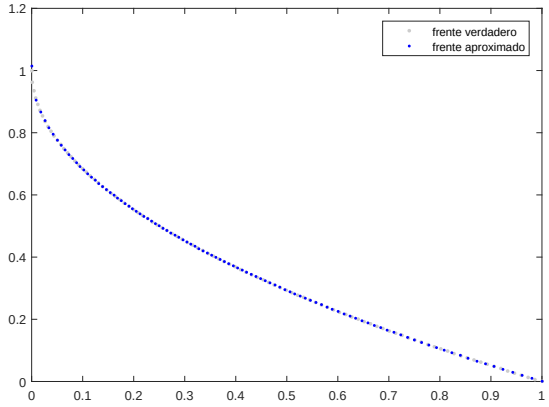
(d) mocDE

Figura C-9: Frentes de Pareto para el problema de prueba Murata

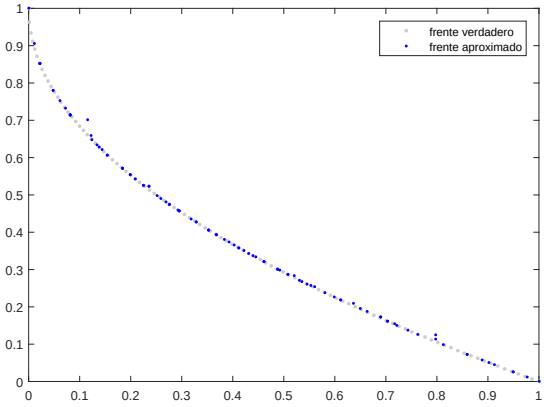


(a) compactMOPSO

(b) SMPSO

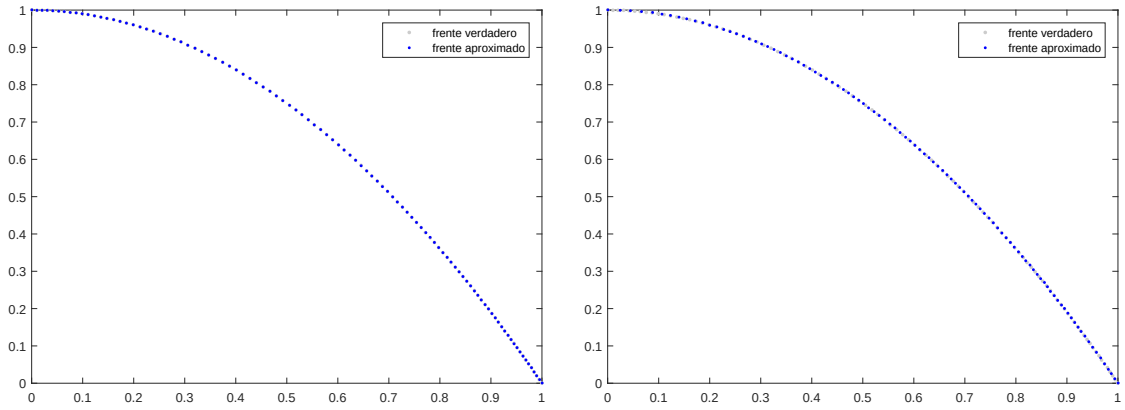


(c) dMOPSO



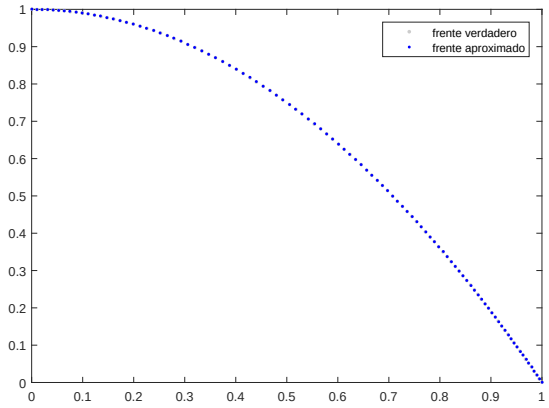
(d) mocDE

Figura C-10: Frentes de Pareto para el problema de prueba ZDT1

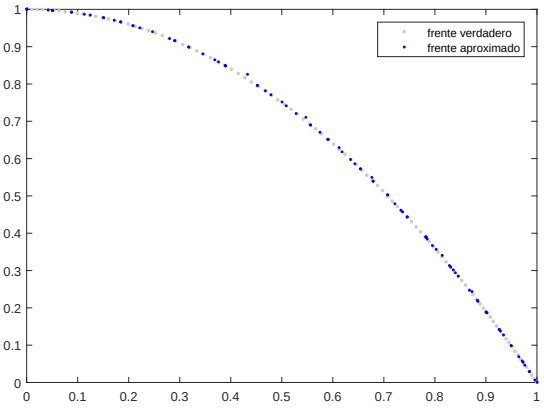


(a) compactMOPSO

(b) SMPSO

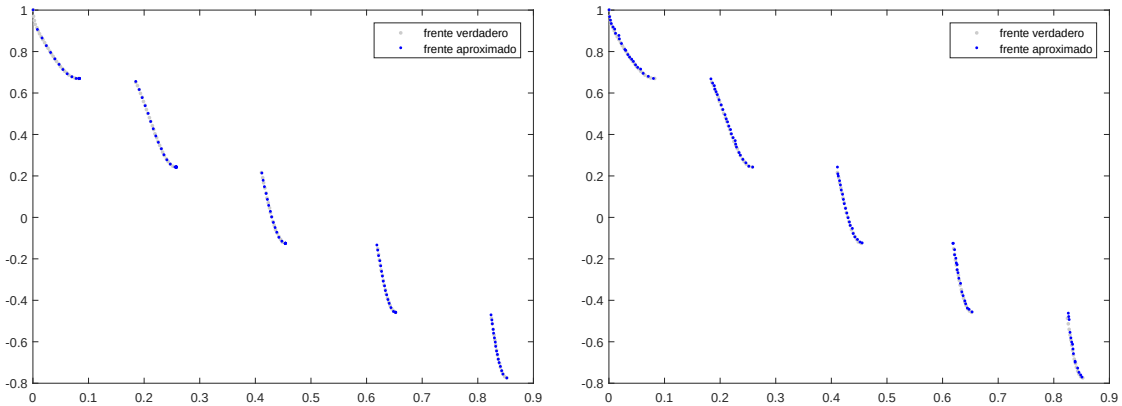


(c) dMOPSO



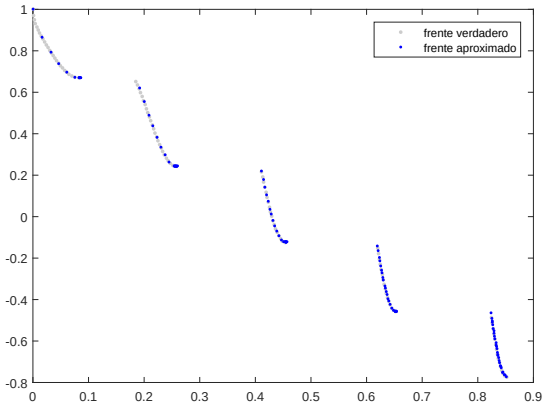
(d) mocDE

Figura C-11: Frentes de Pareto para el problema de prueba ZDT2

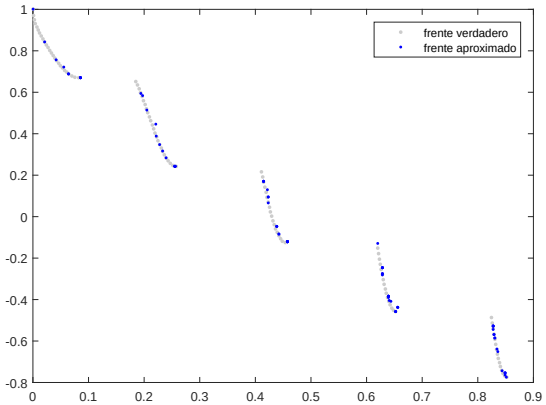


(a) compactMOPSO

(b) SMPSO

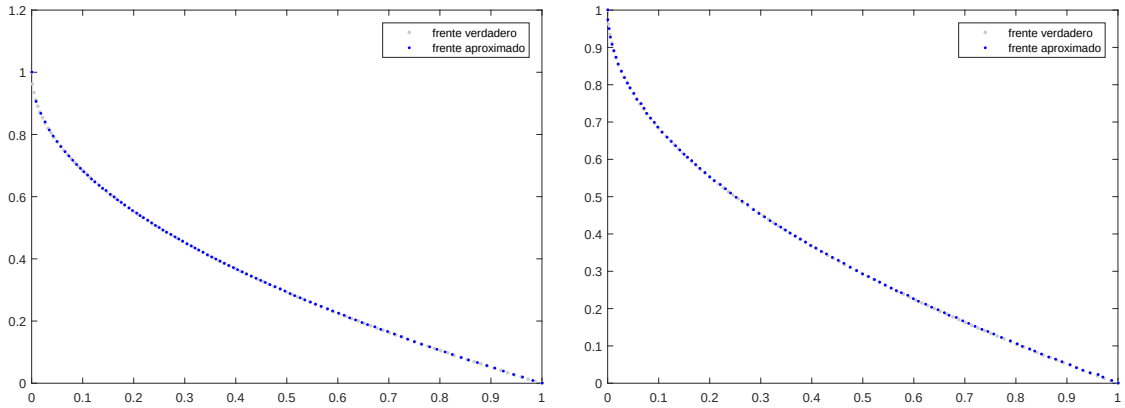


(c) dMOPSO



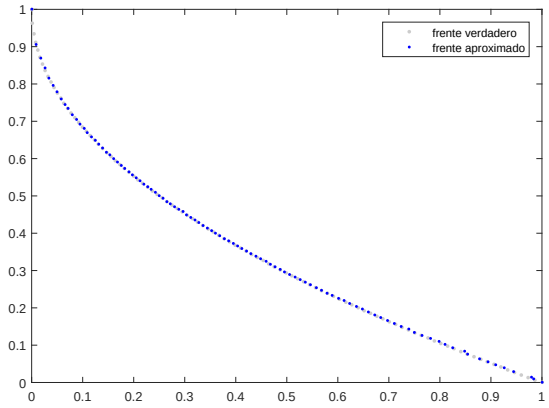
(d) mocDE

Figura C-12: Frentes de Pareto para el problema de prueba ZDT3

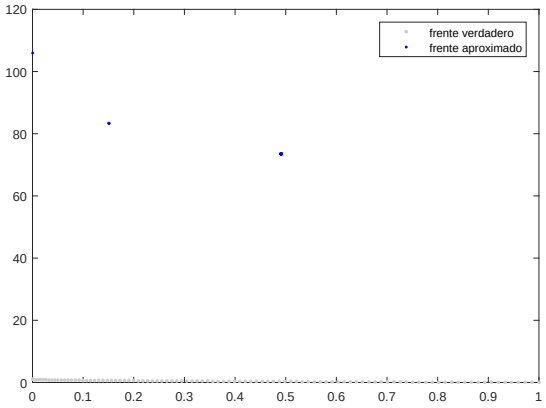


(a) compactMOPSO

(b) SMPSO

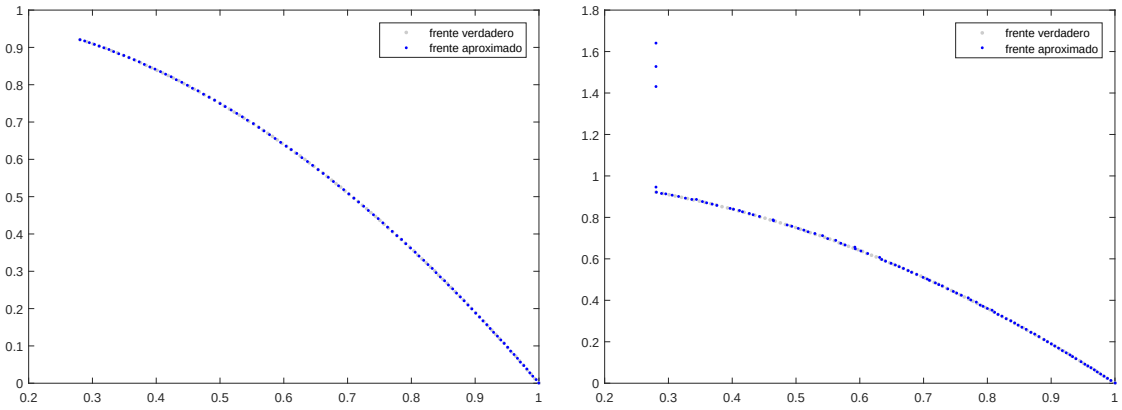


(c) dMOPSO



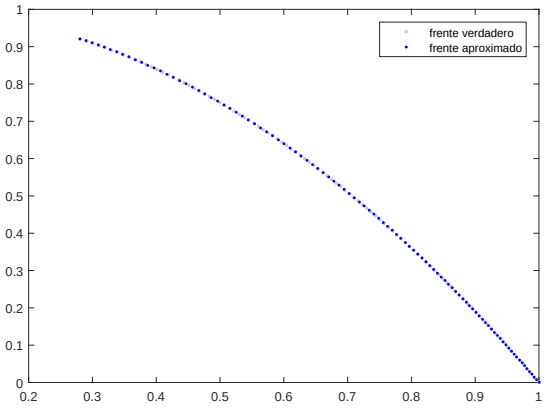
(d) mocDE

Figura C-13: Frentes de Pareto para el problema de prueba ZDT4

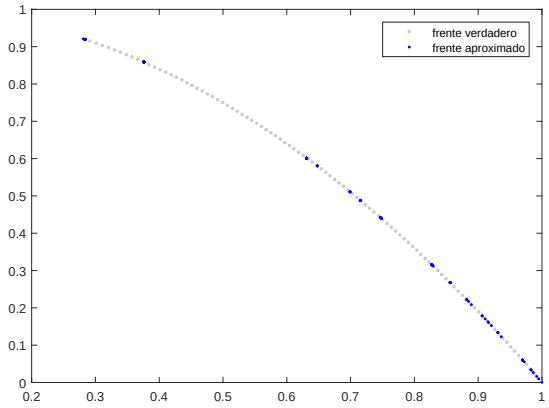


(a) compactMOPSO

(b) SMPSO



(c) dMOPSO



(d) mocDE

Figura C-14: Frentes de Pareto para el problema de prueba ZDT6

C.2. Resultados gráficos de los indicadores

C.2.1. Optimizadores multi-objetivo basados en cúmulos de partículas (MOPSOs, por sus siglas en inglés)

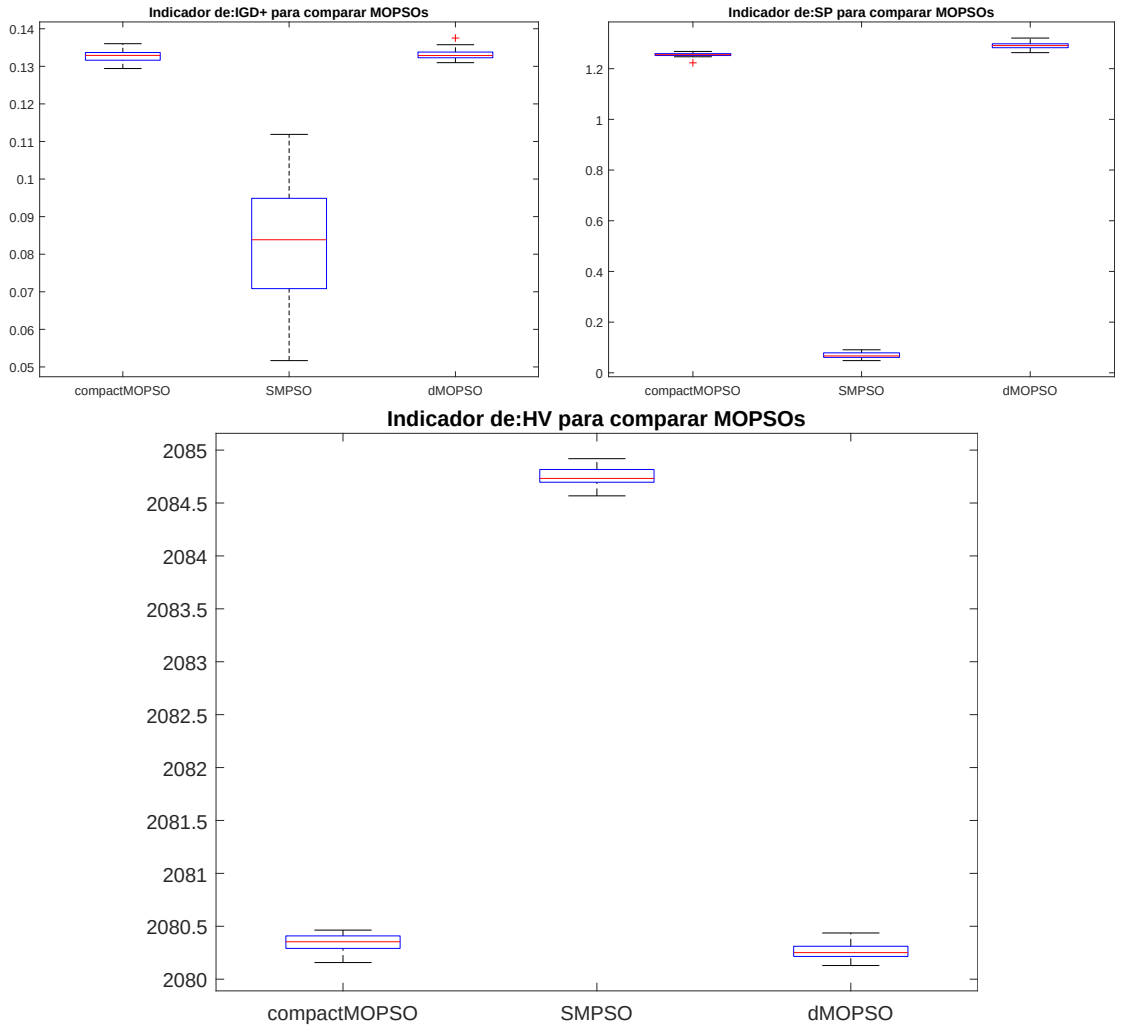


Figura C-15: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Binh

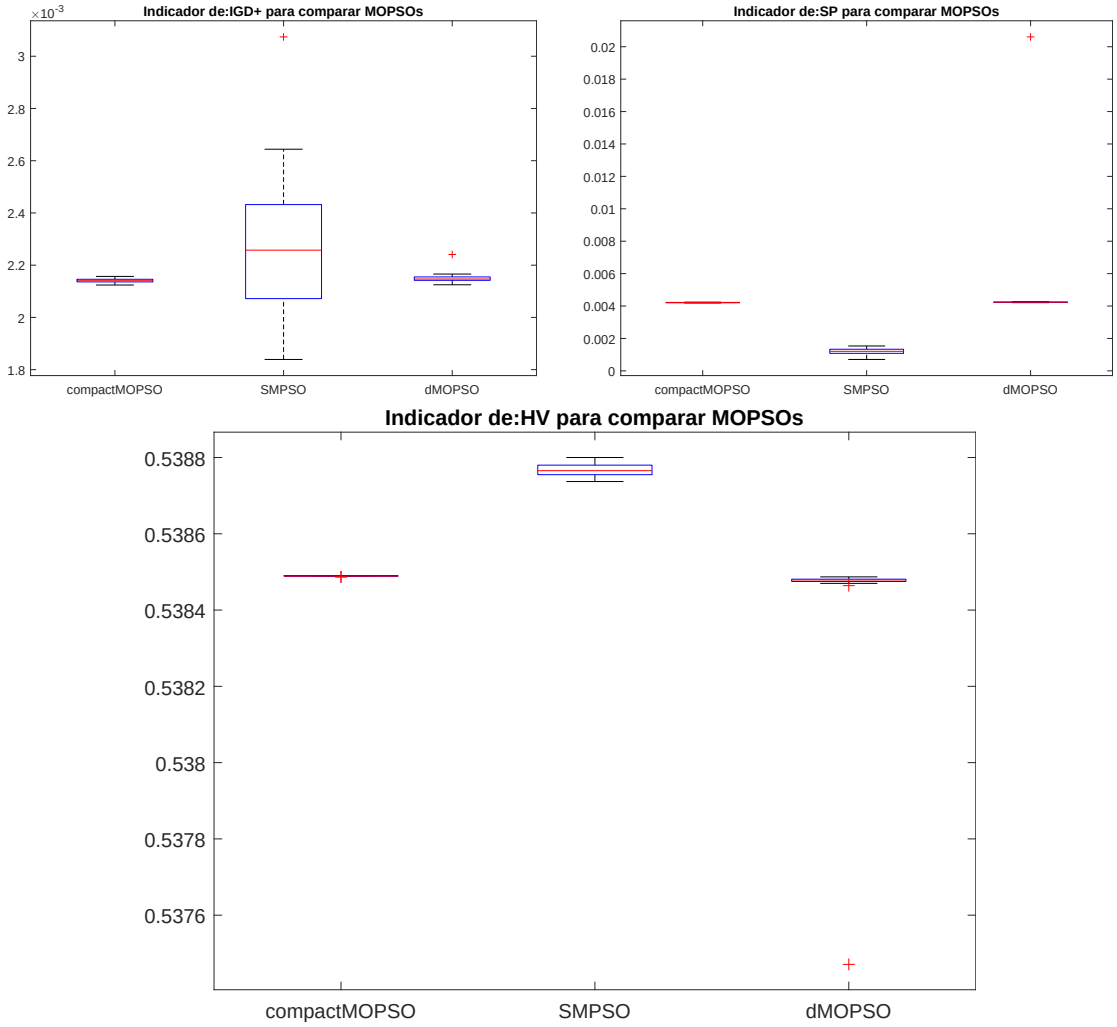


Figura C-16: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Deb1

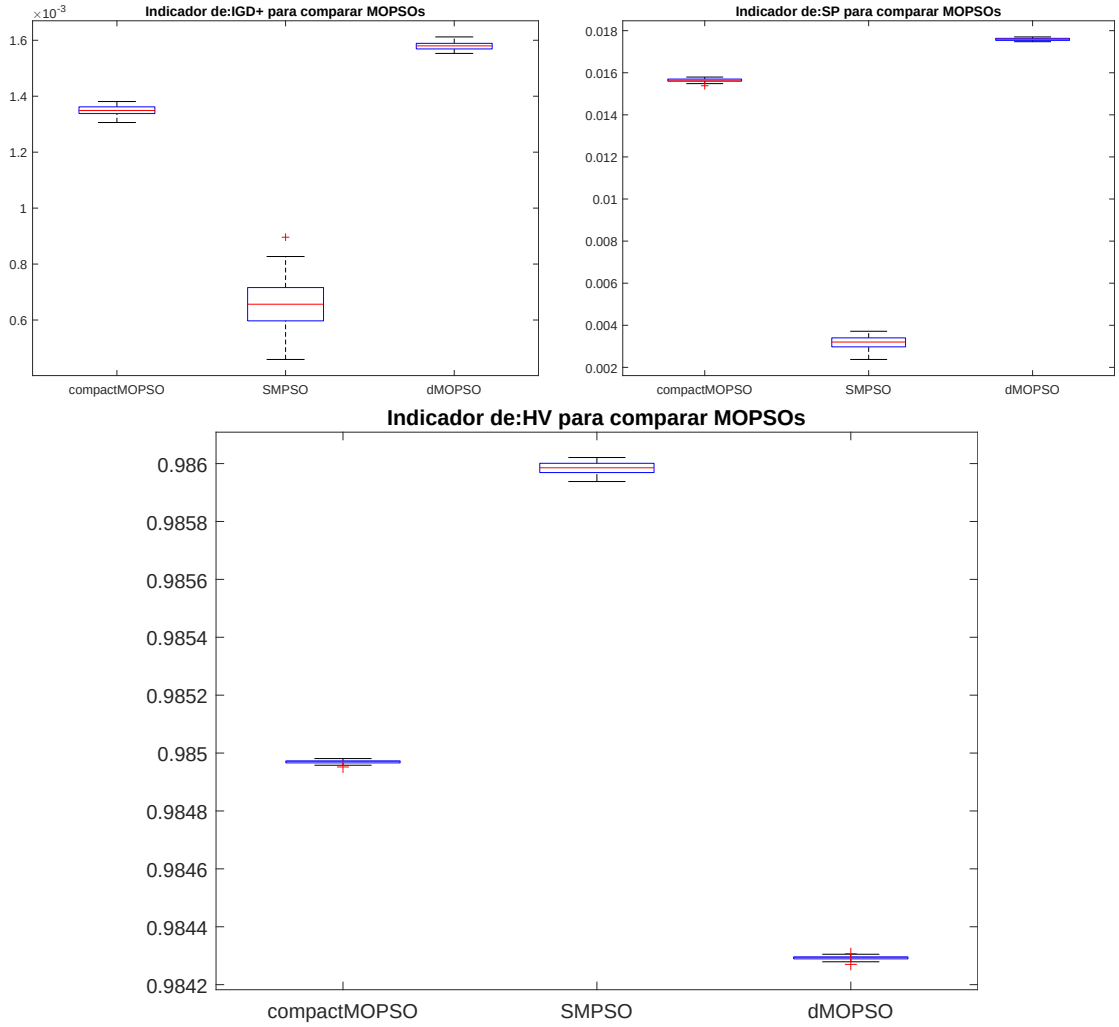


Figura C-17: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Deb2

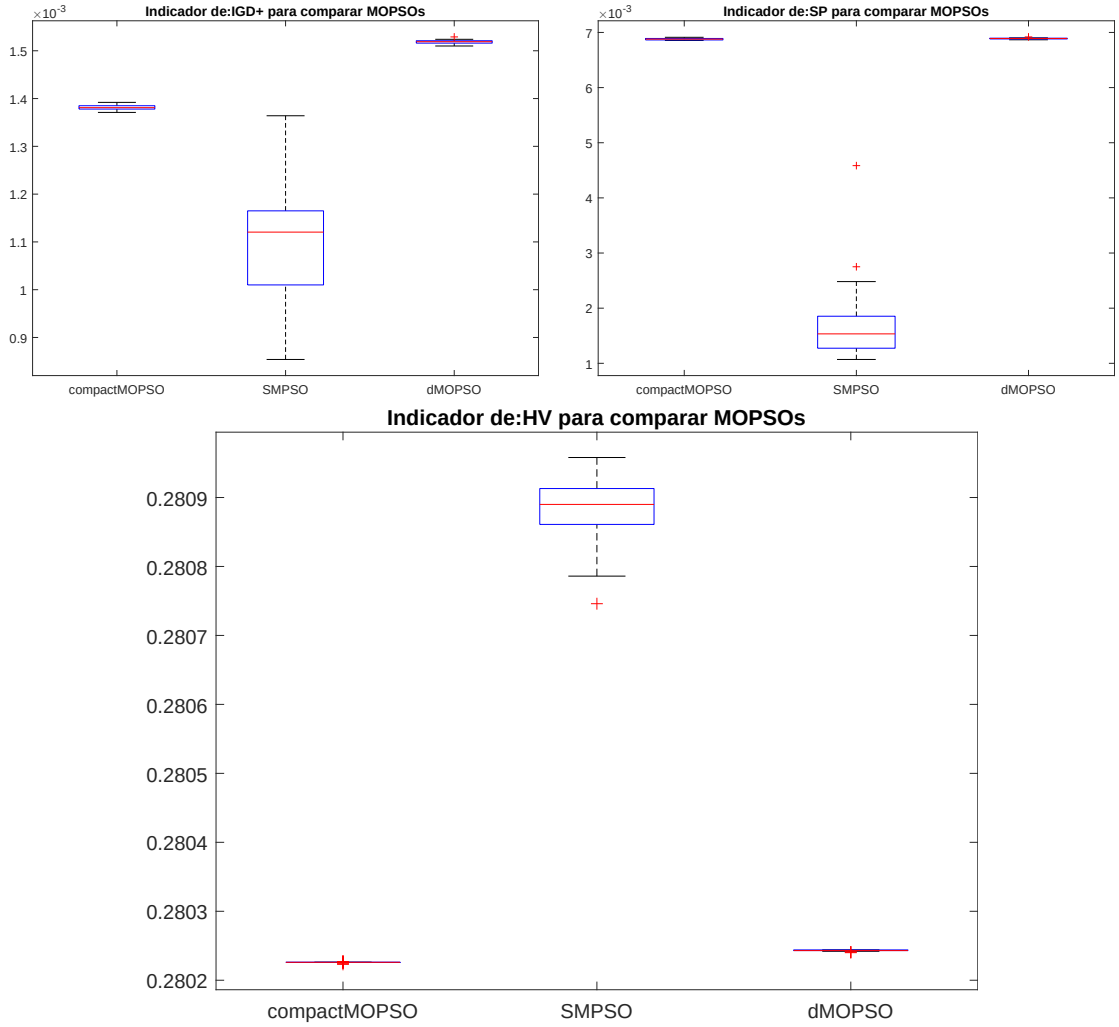


Figura C-18: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Deb3

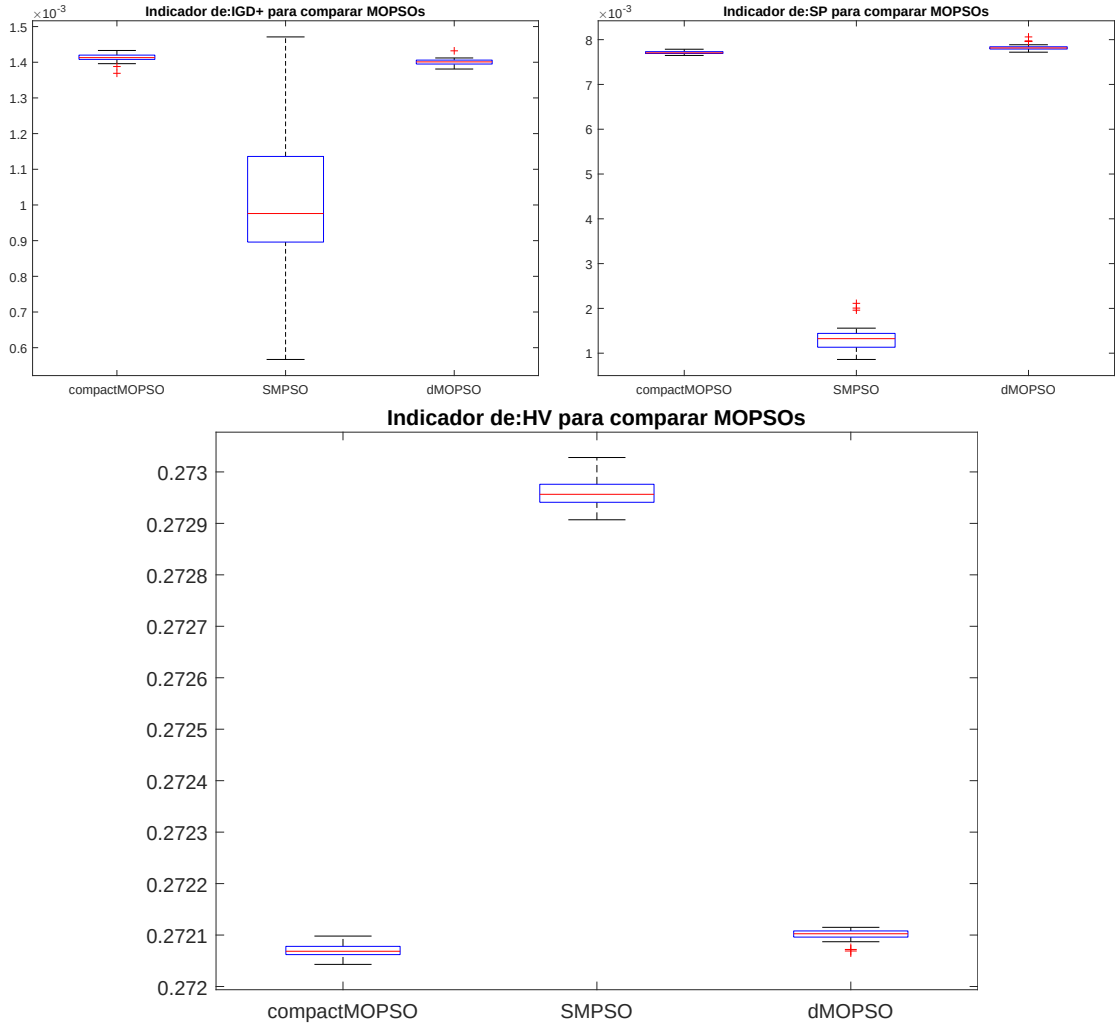


Figura C-19: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Fonseca1

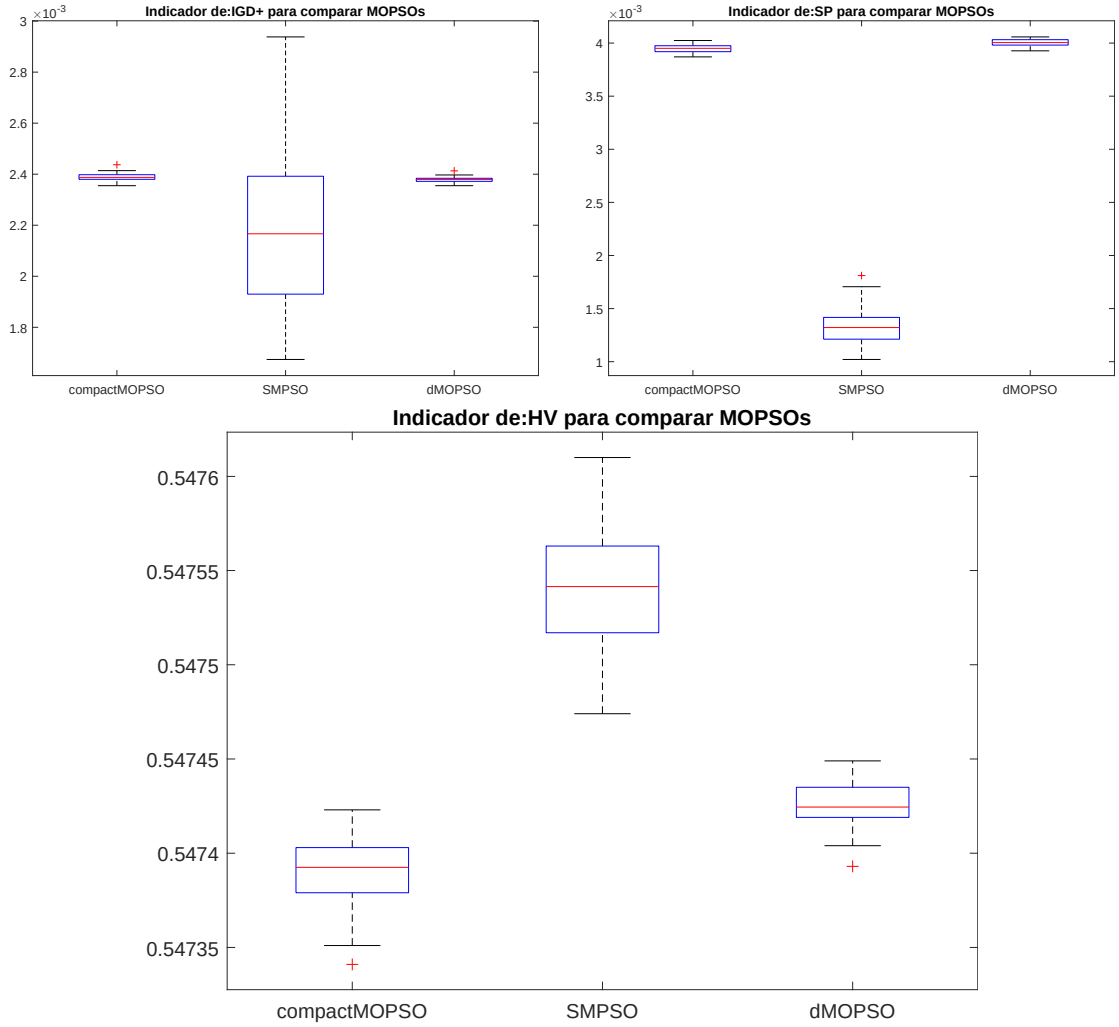


Figura C-20: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Fonseca2

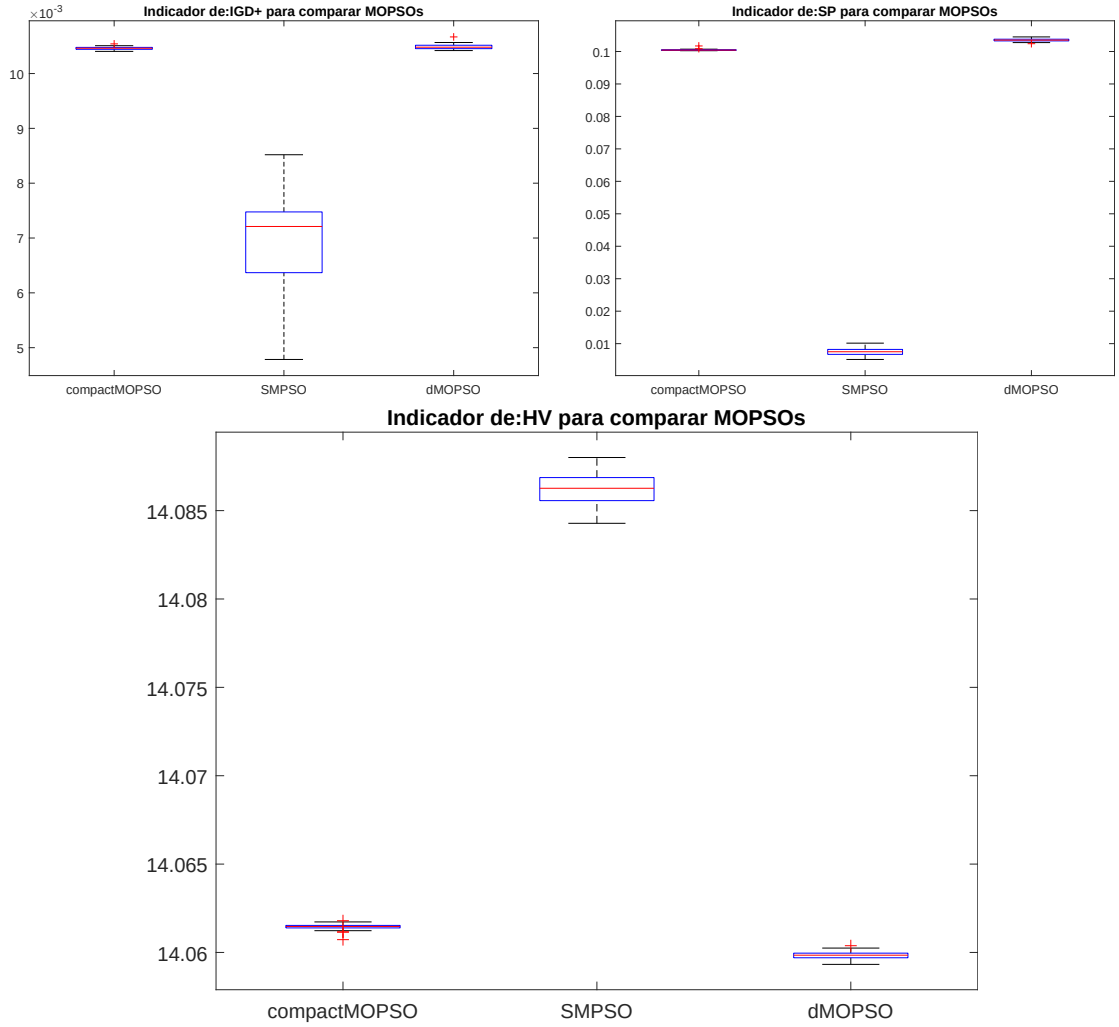


Figura C-21: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Laumanns

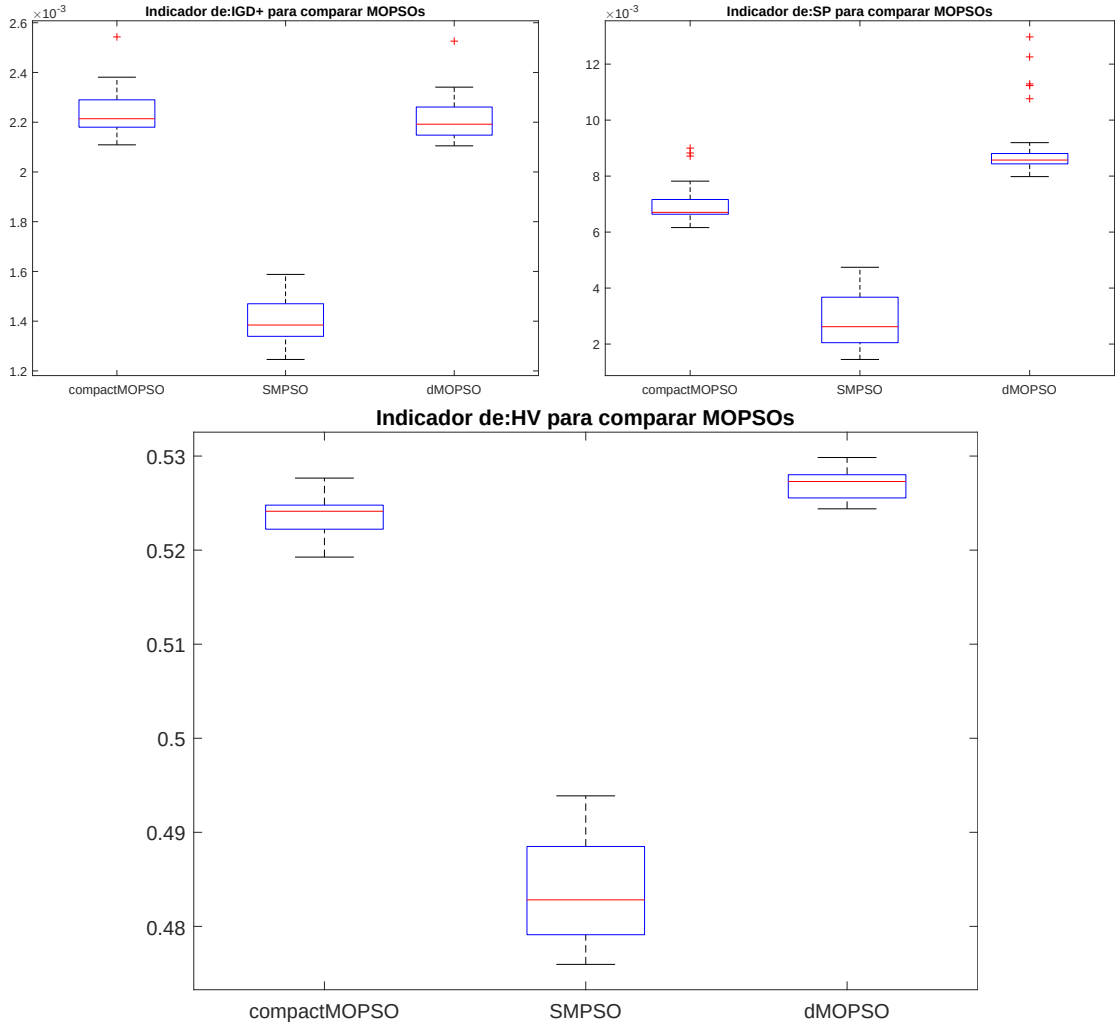


Figura C-22: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Lis

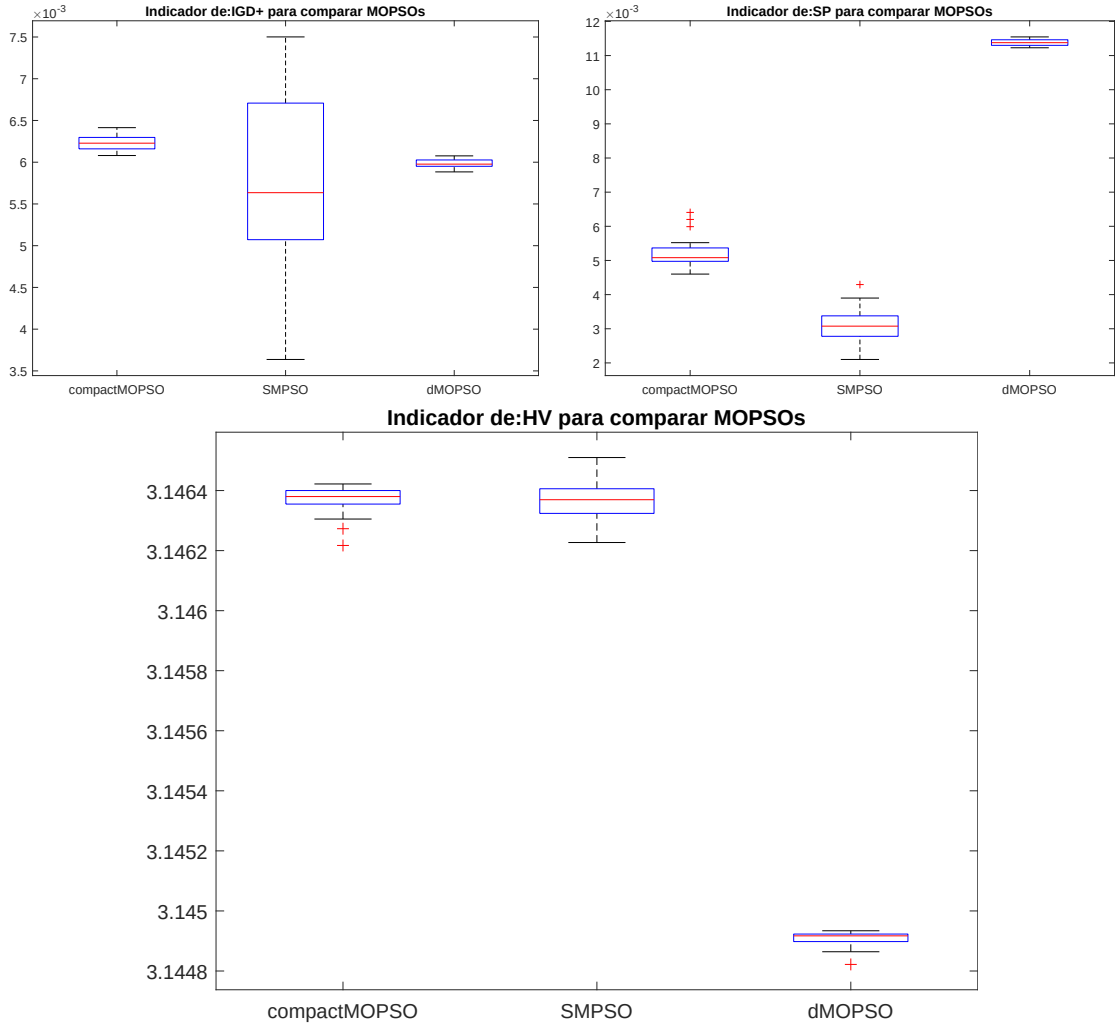


Figura C-23: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Murata

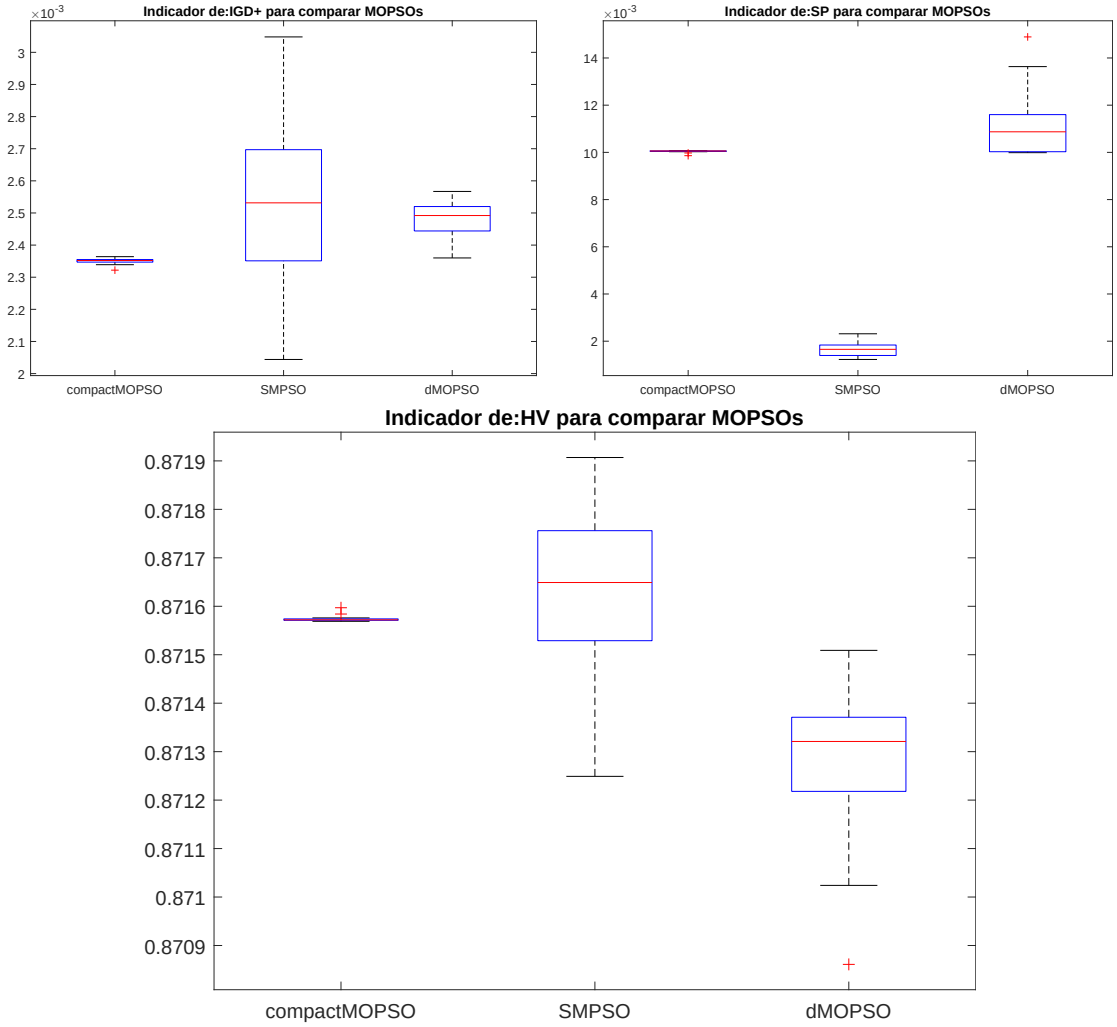


Figura C-24: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT1

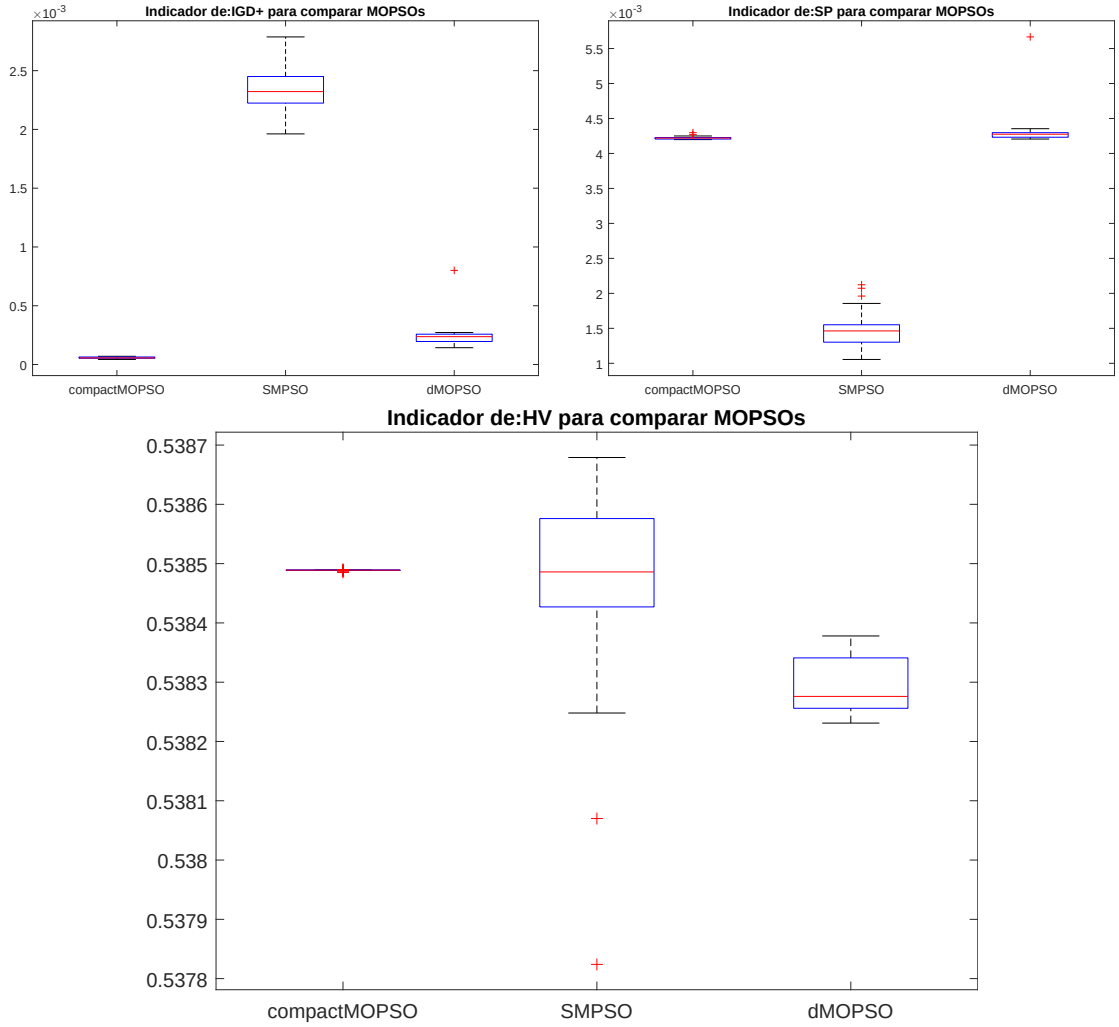


Figura C-25: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT2

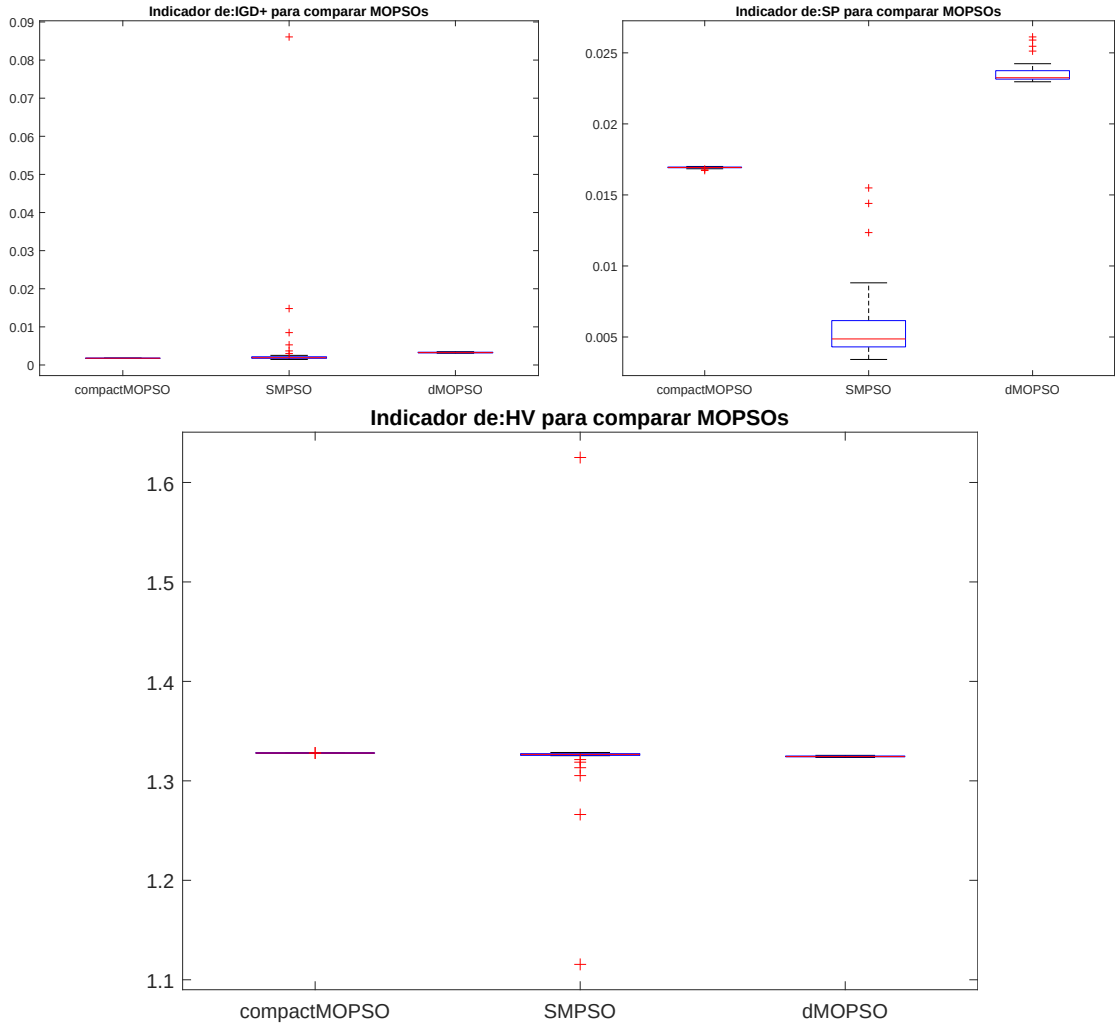


Figura C-26: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT3

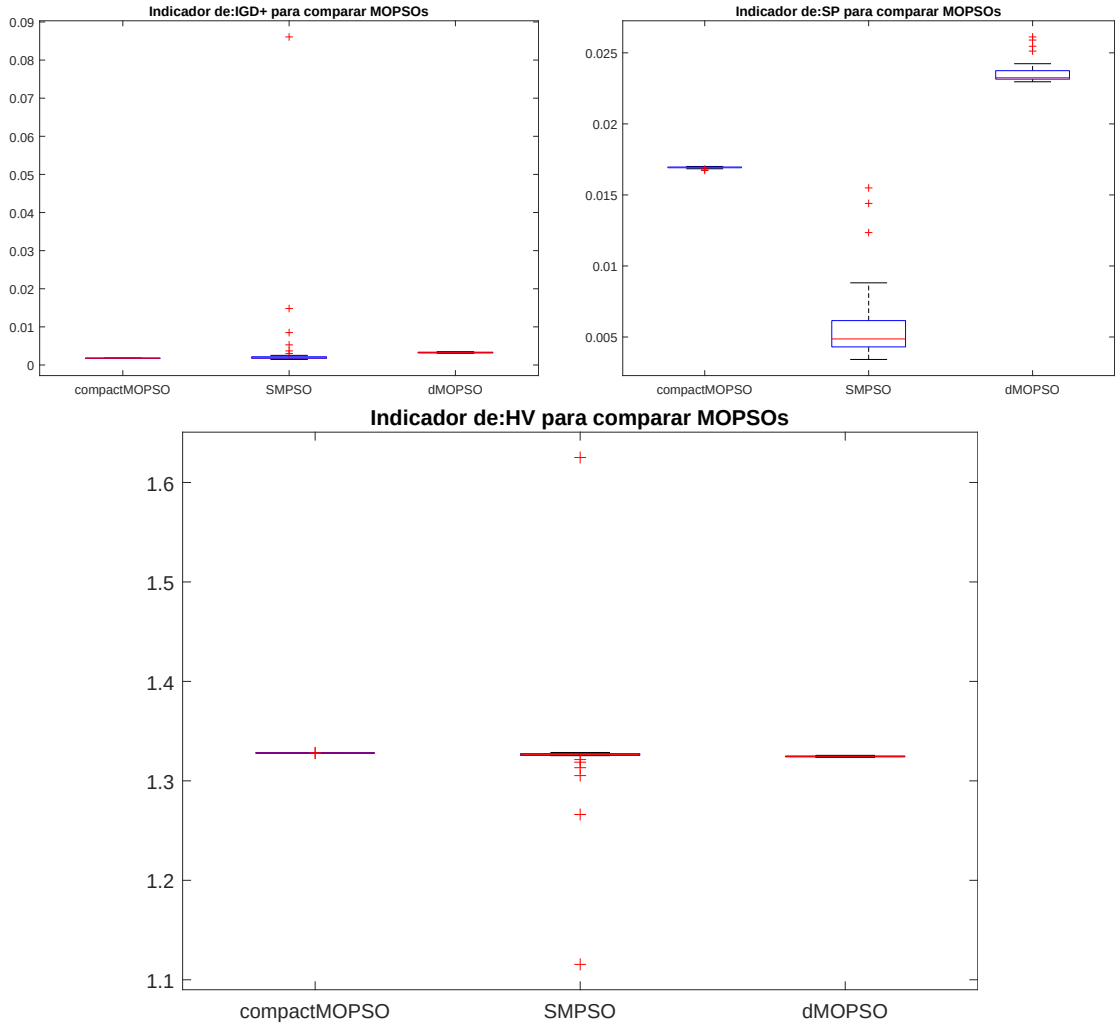


Figura C-27: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT3

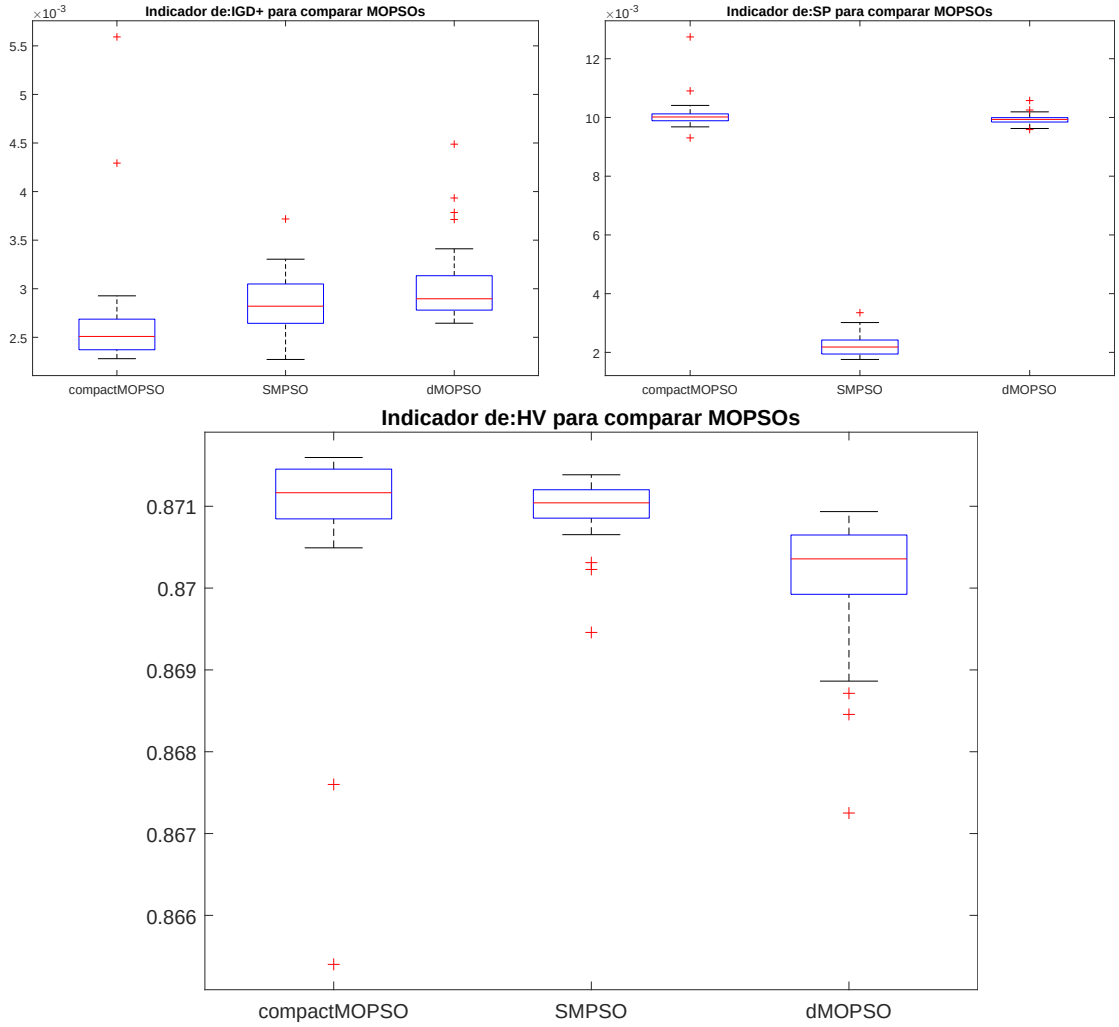


Figura C-28: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT4

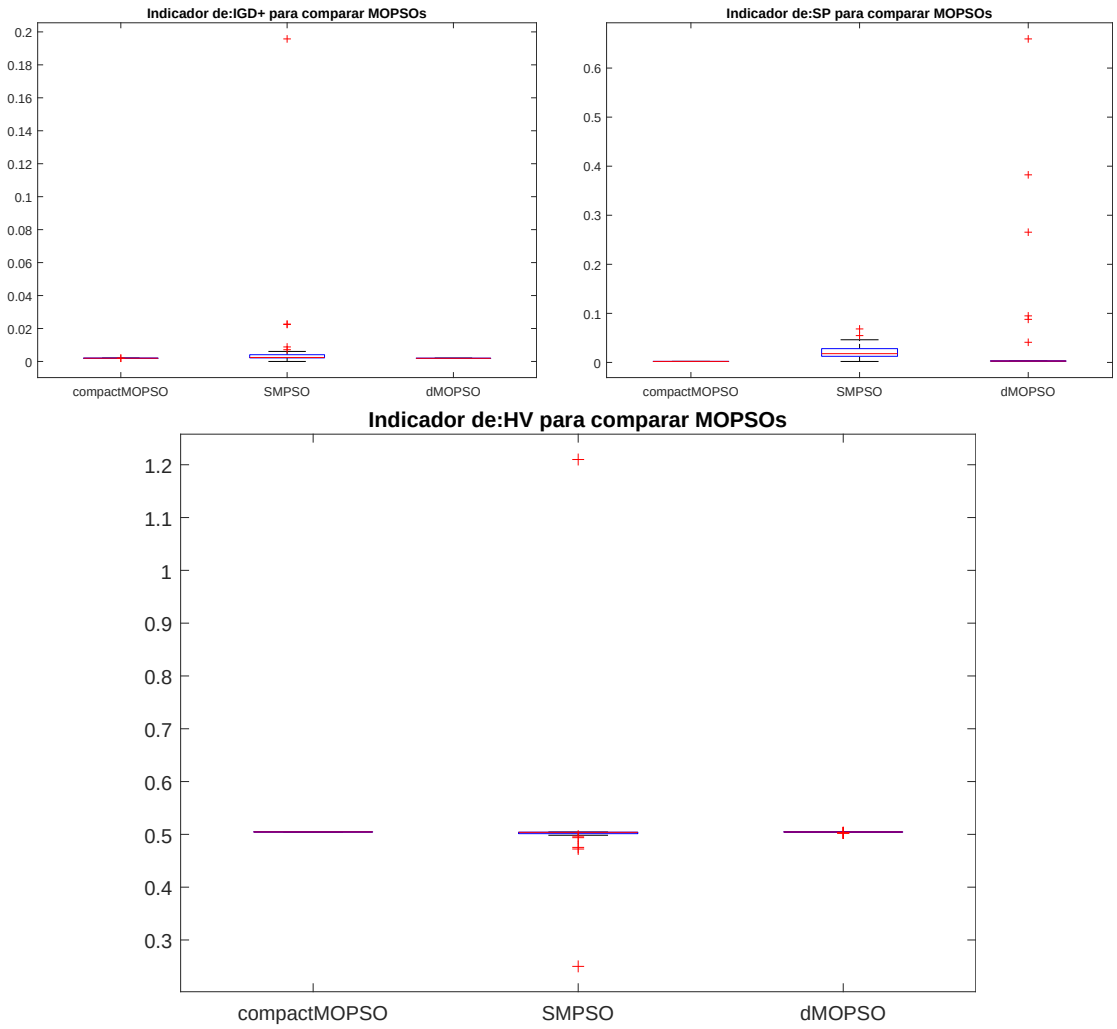


Figura C-29: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT6

C.2.2. AEMOs

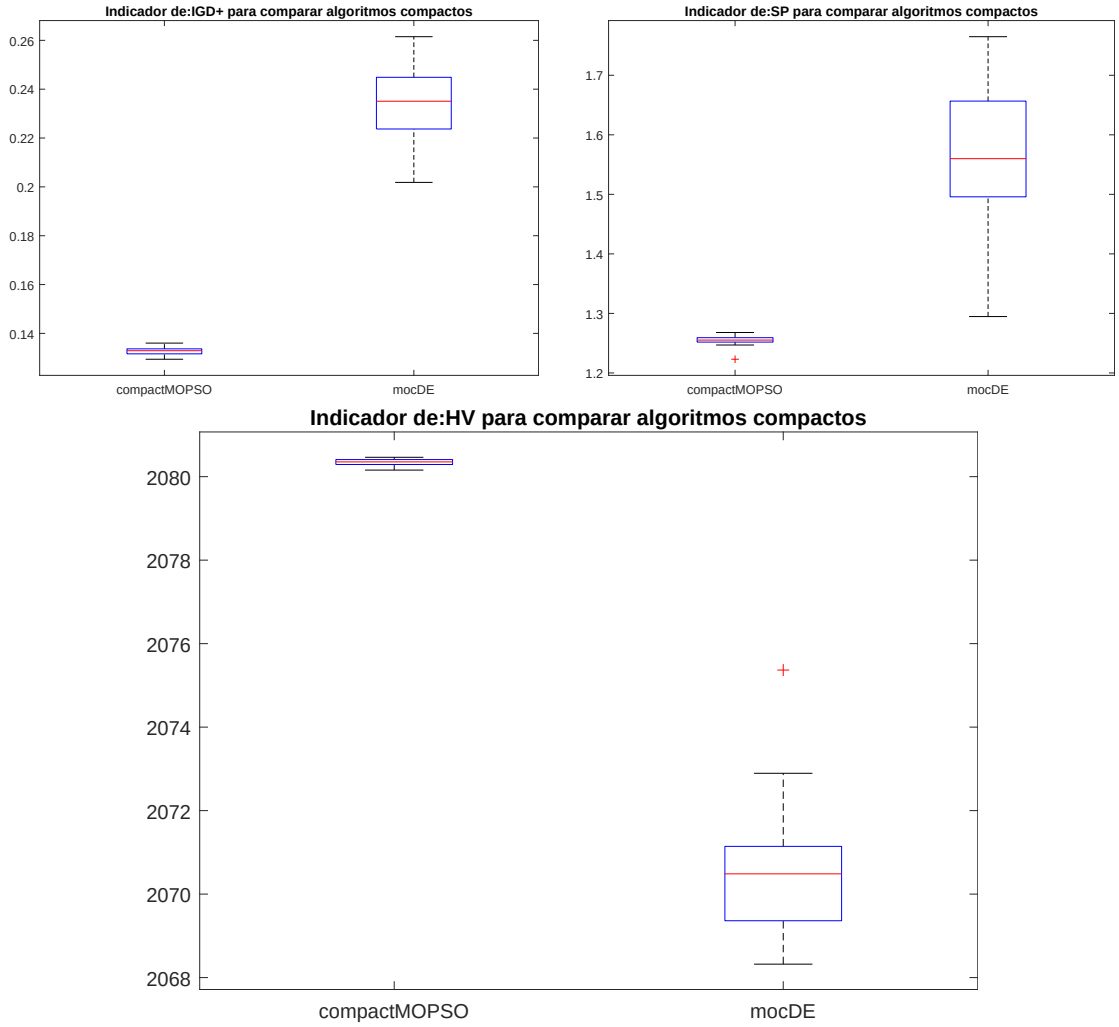


Figura C-30: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Binh

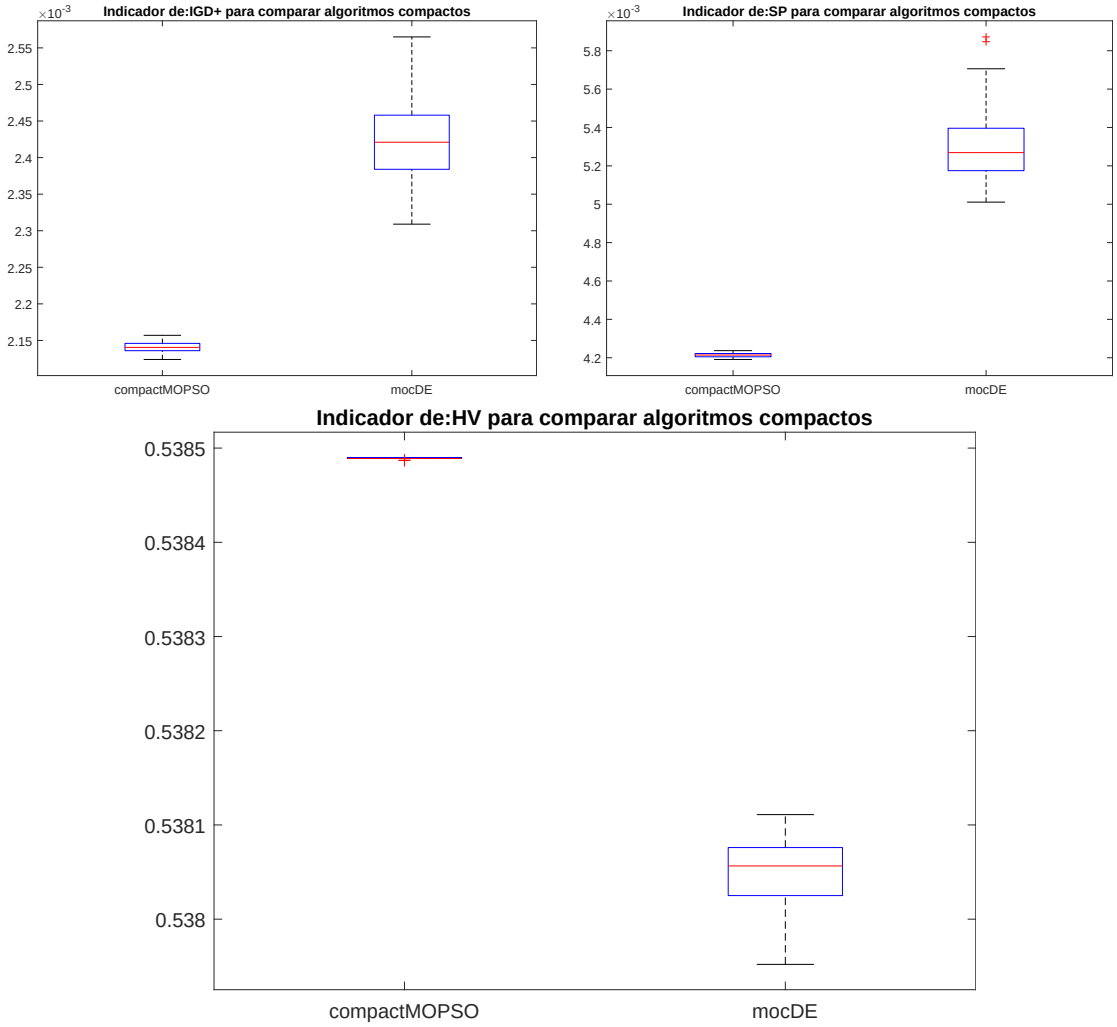


Figura C-31: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Deb1

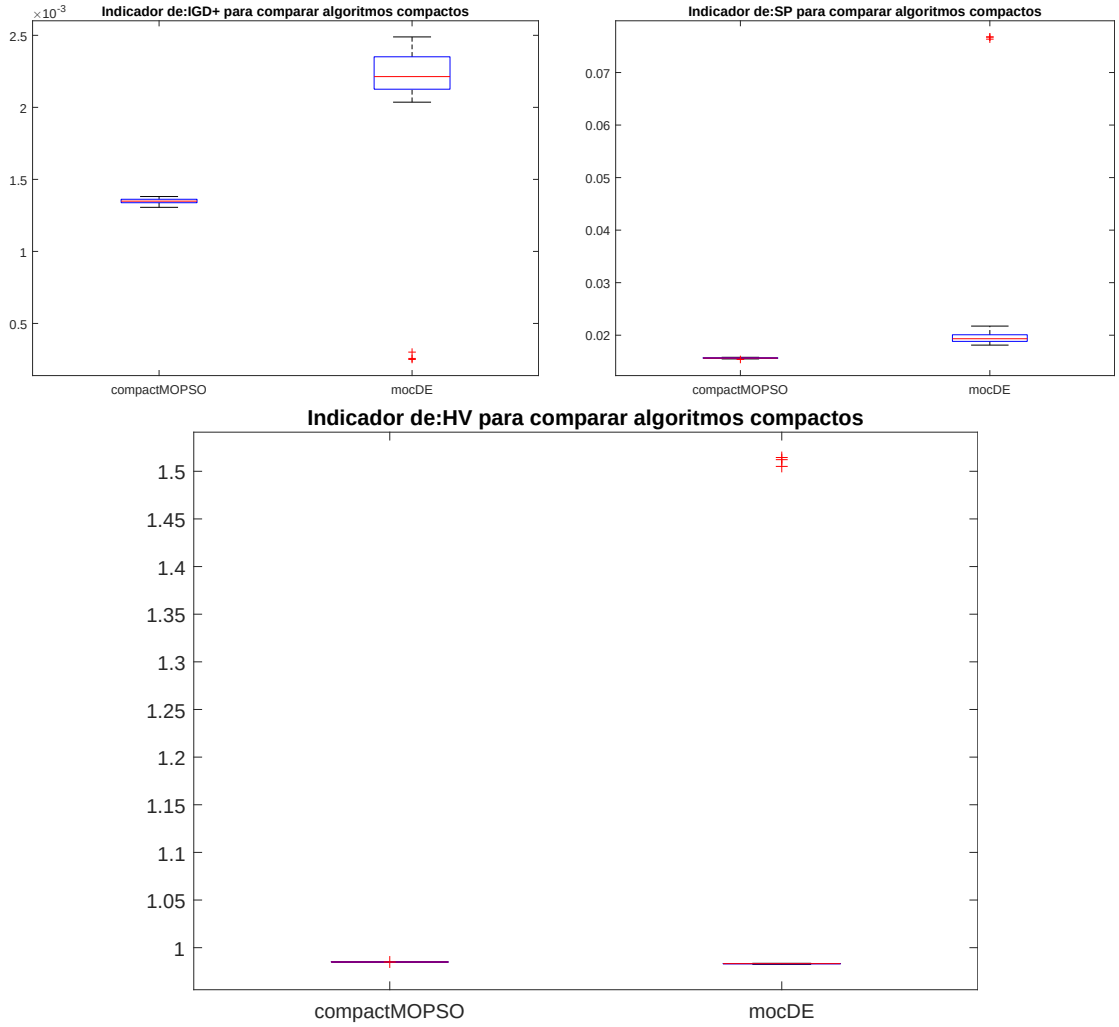


Figura C-32: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Deb2

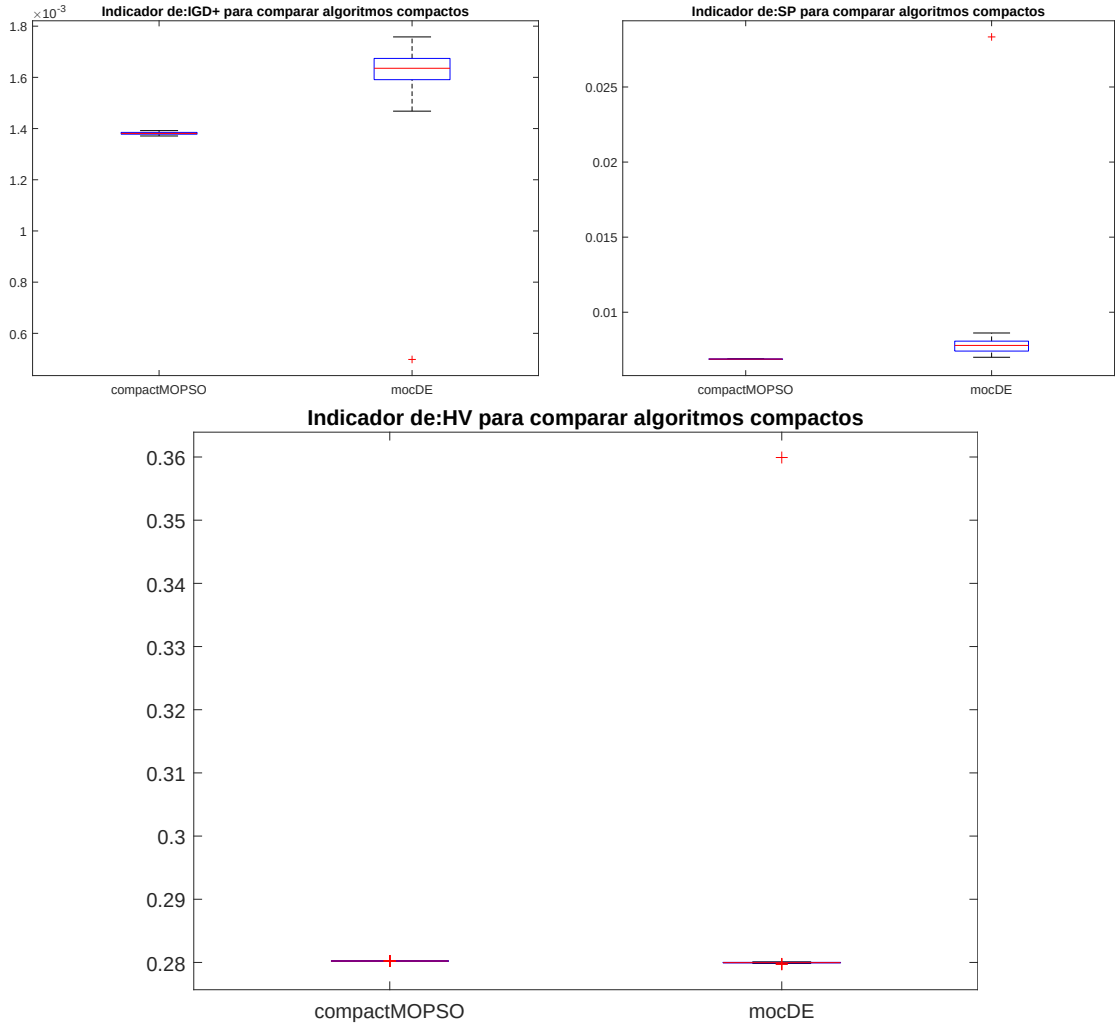


Figura C-33: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Deb3

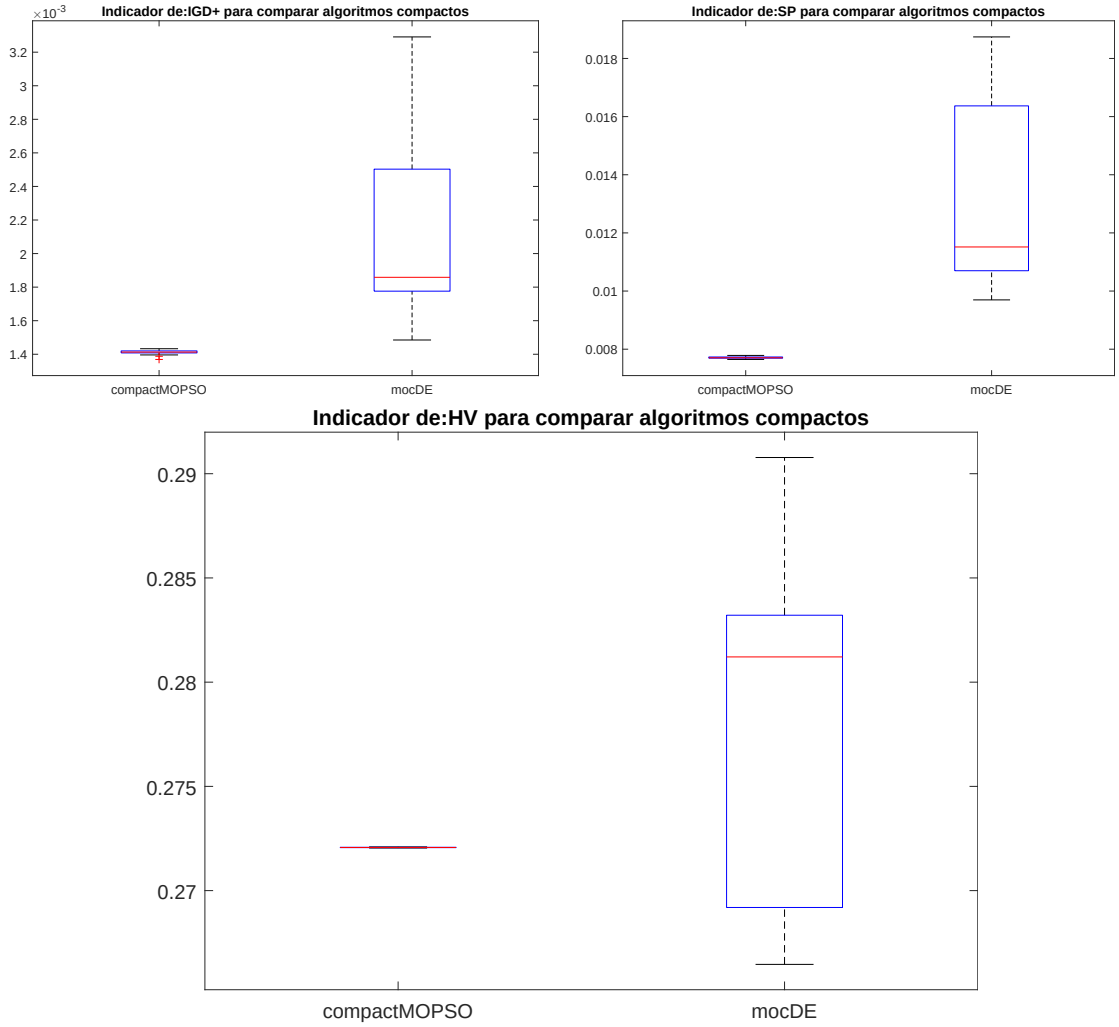


Figura C-34: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Fonseca1

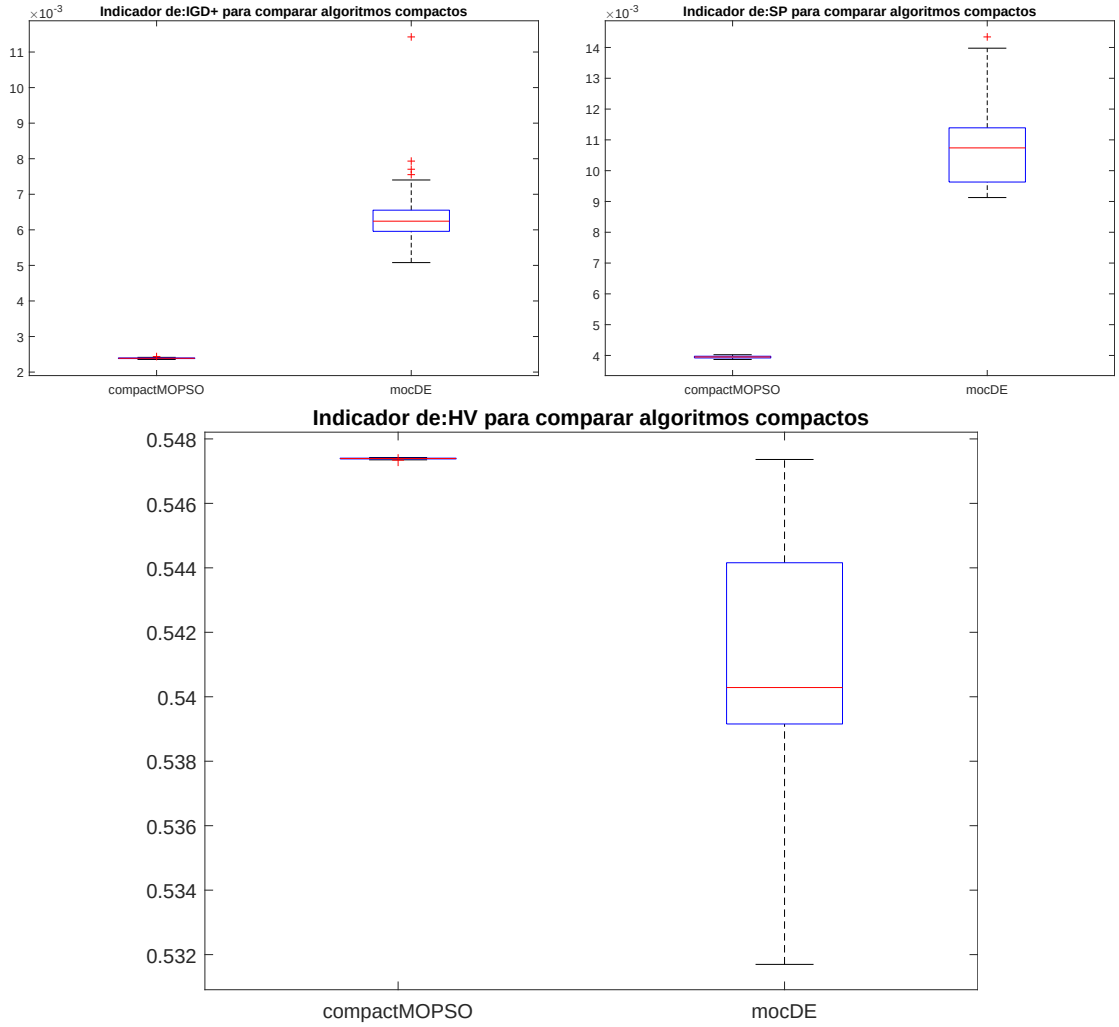


Figura C-35: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Fonseca2

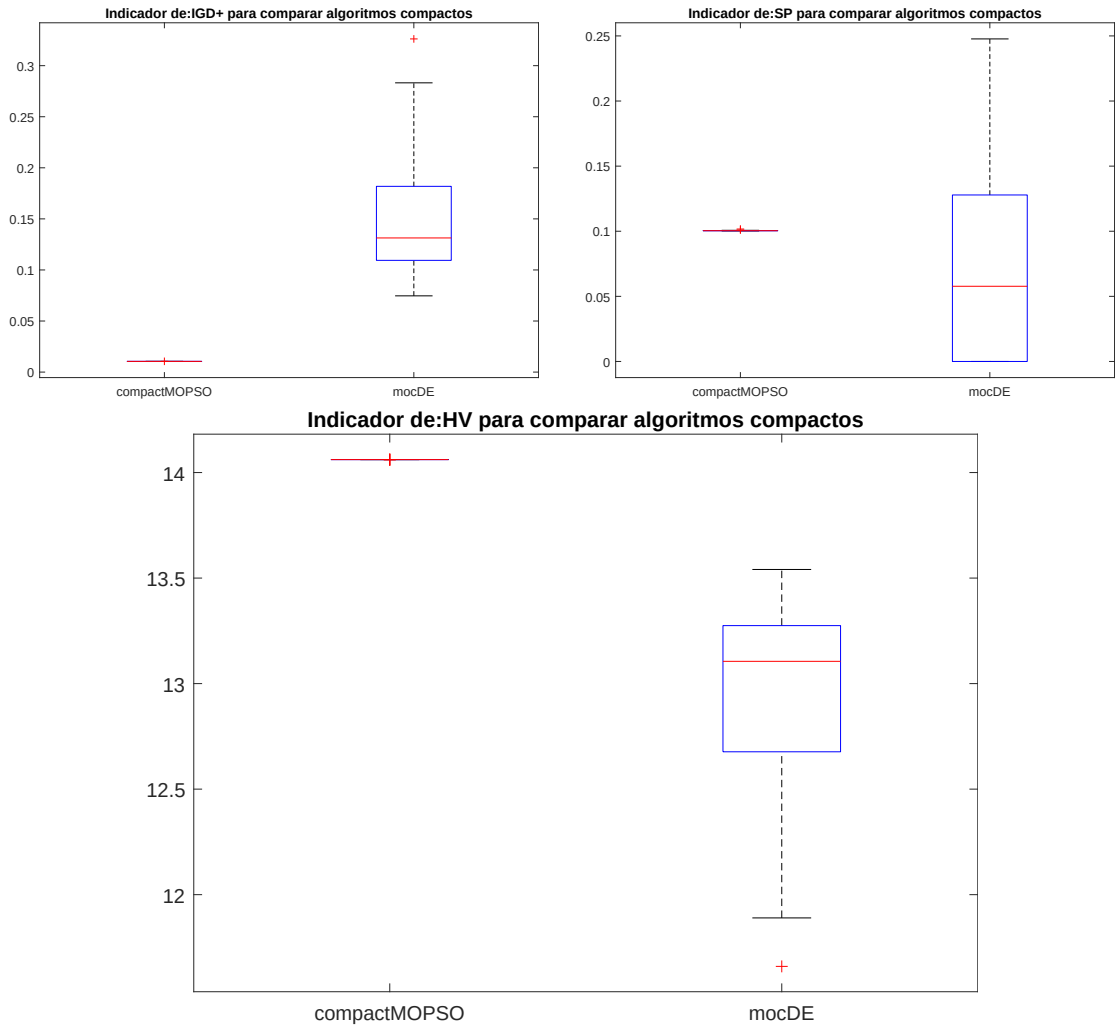


Figura C-36: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Laumanns

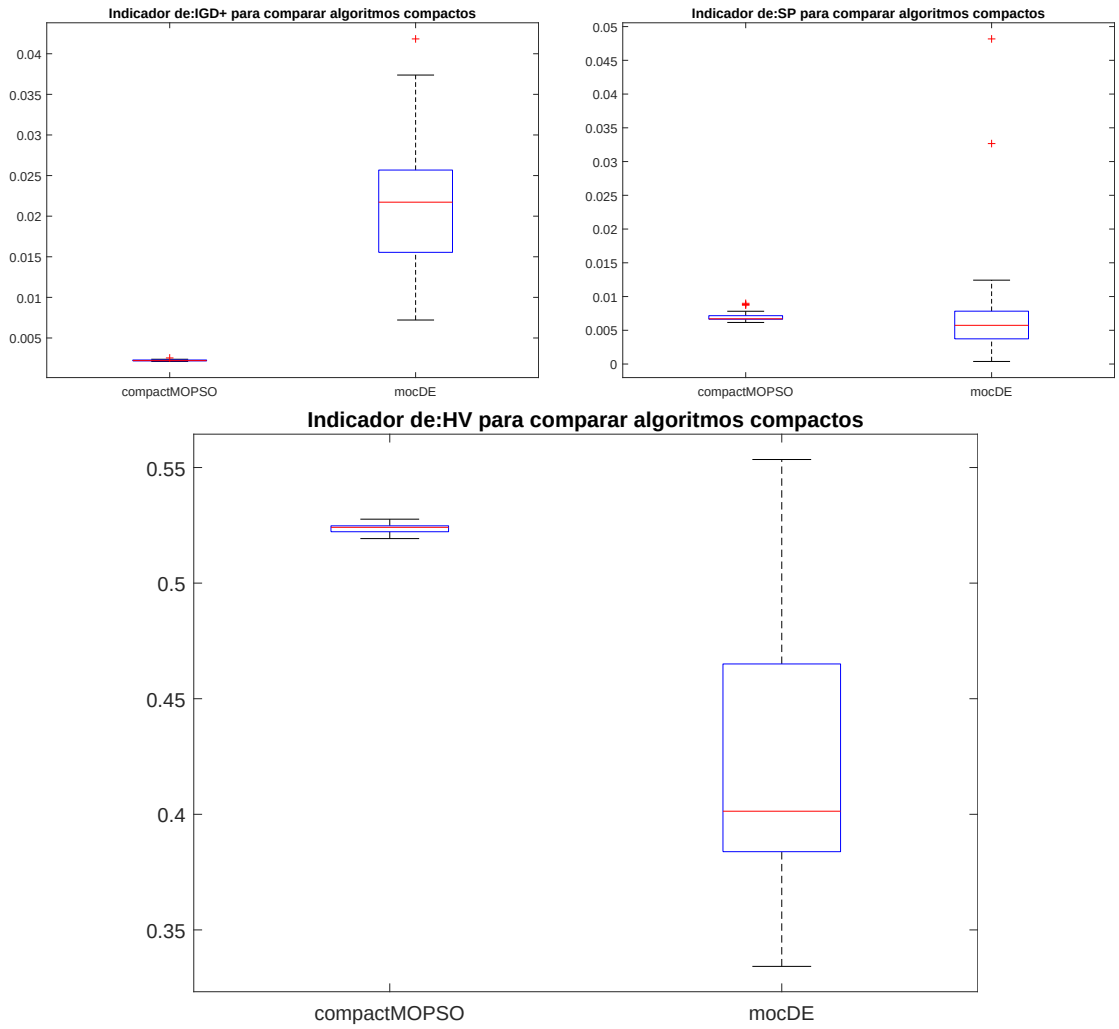


Figura C-37: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Lis

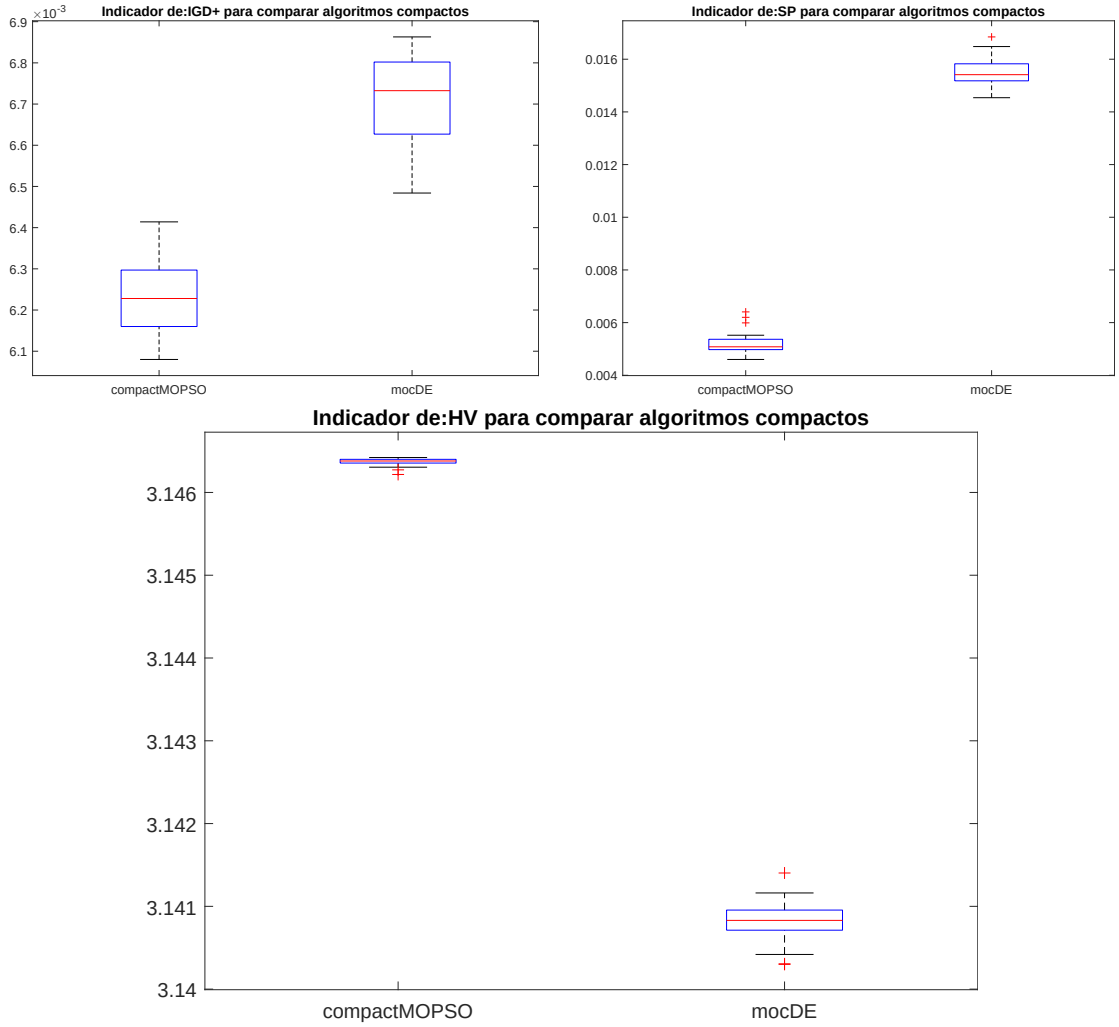


Figura C-38: Gráficas de caja de los tres indicadores utilizados para el problema de prueba Murata

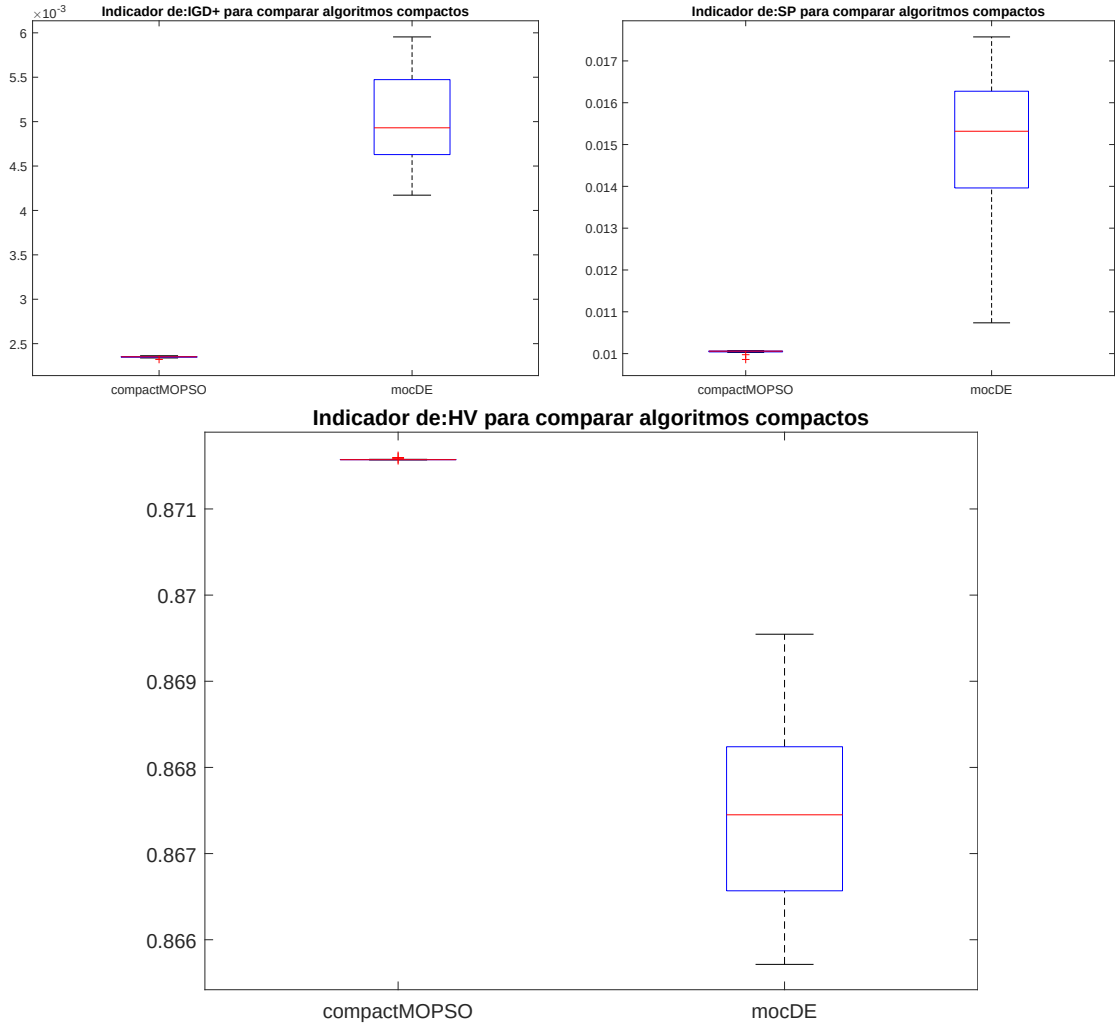


Figura C-39: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT1

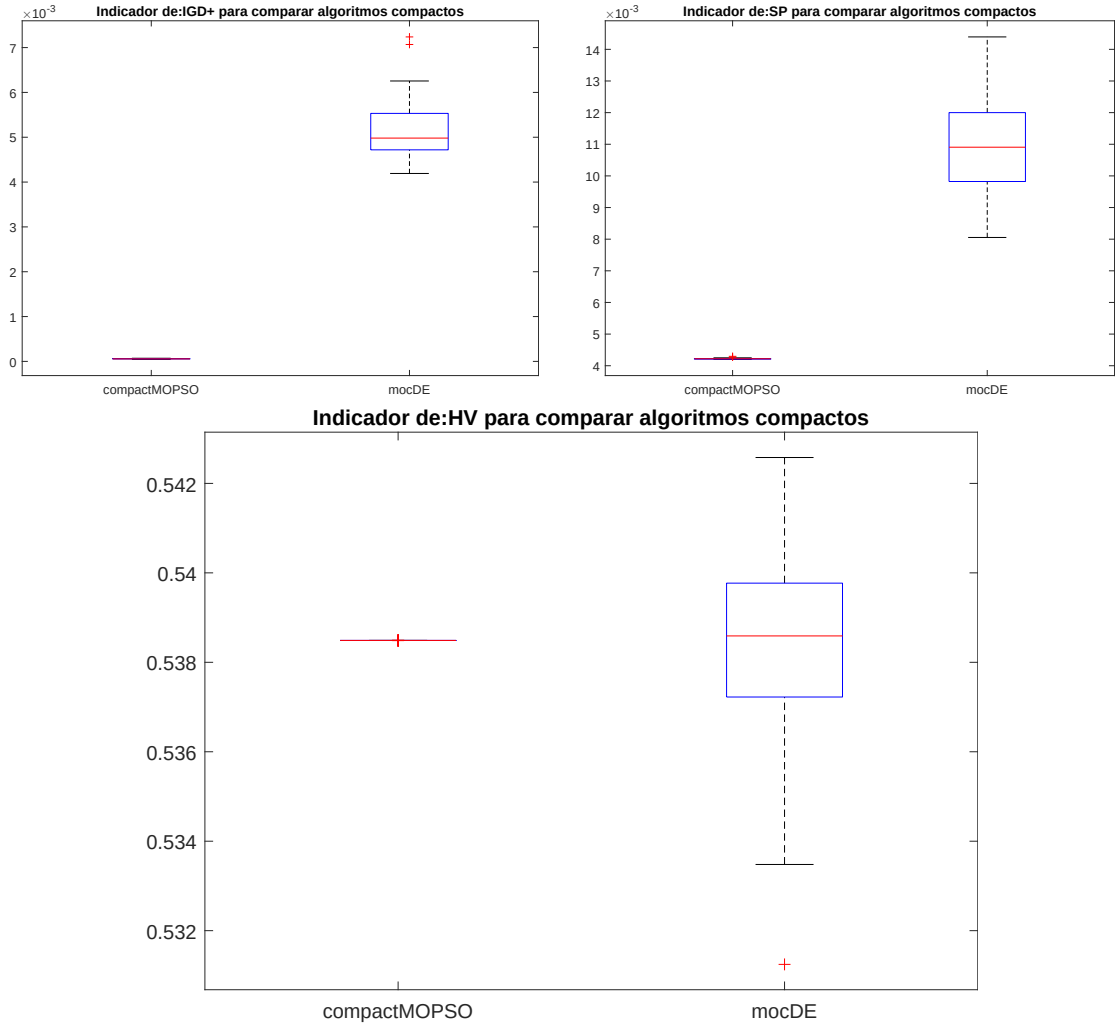


Figura C-40: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT2

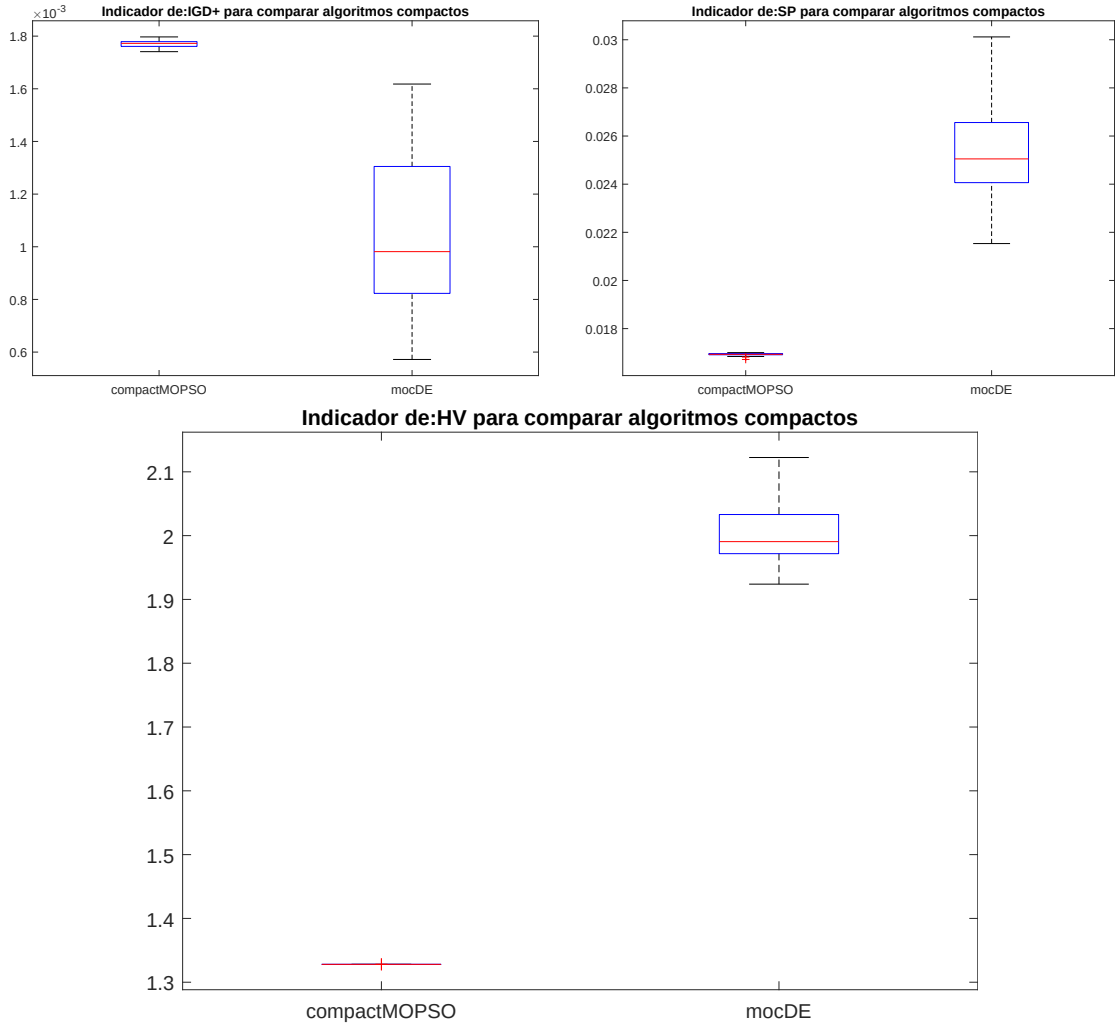


Figura C-41: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT3

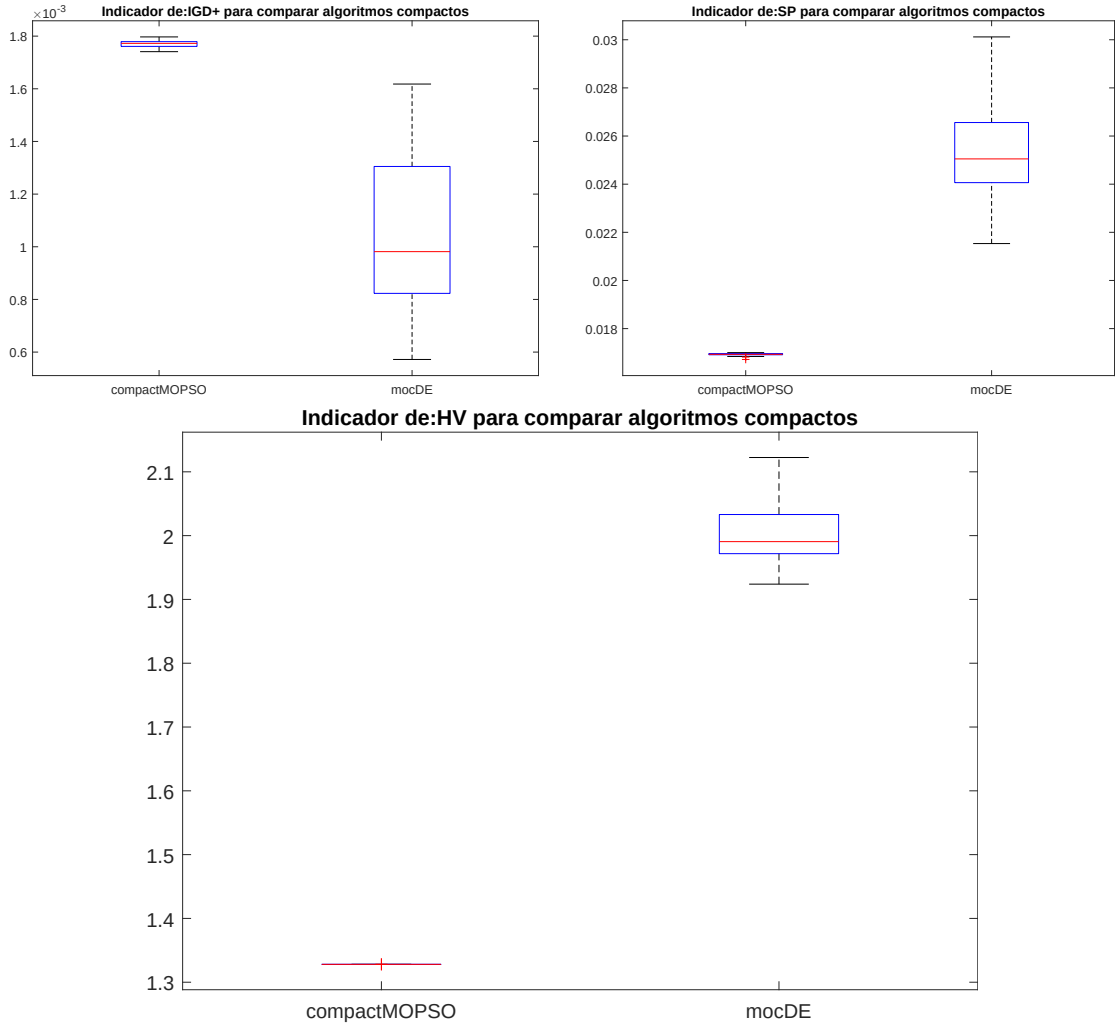


Figura C-42: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT3

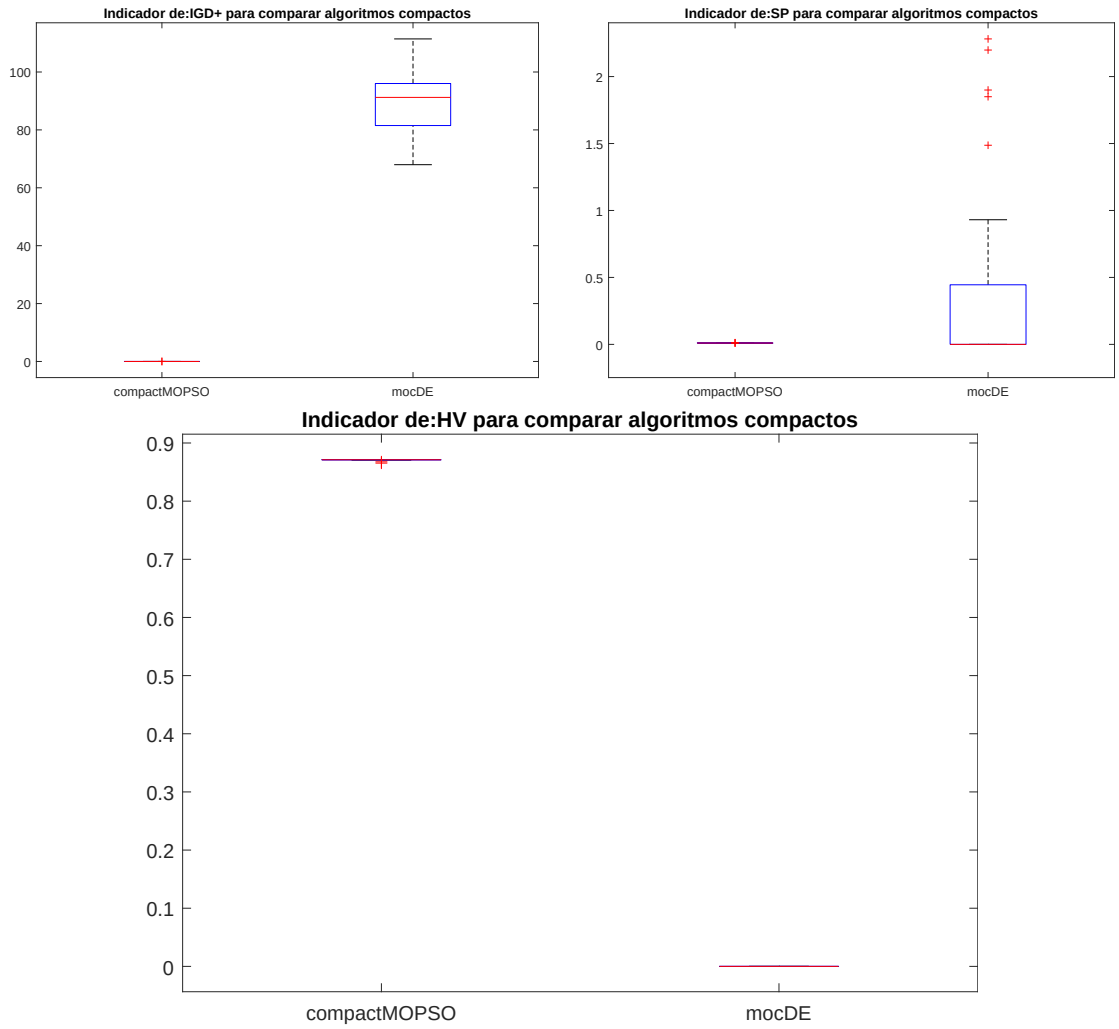


Figura C-43: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT4

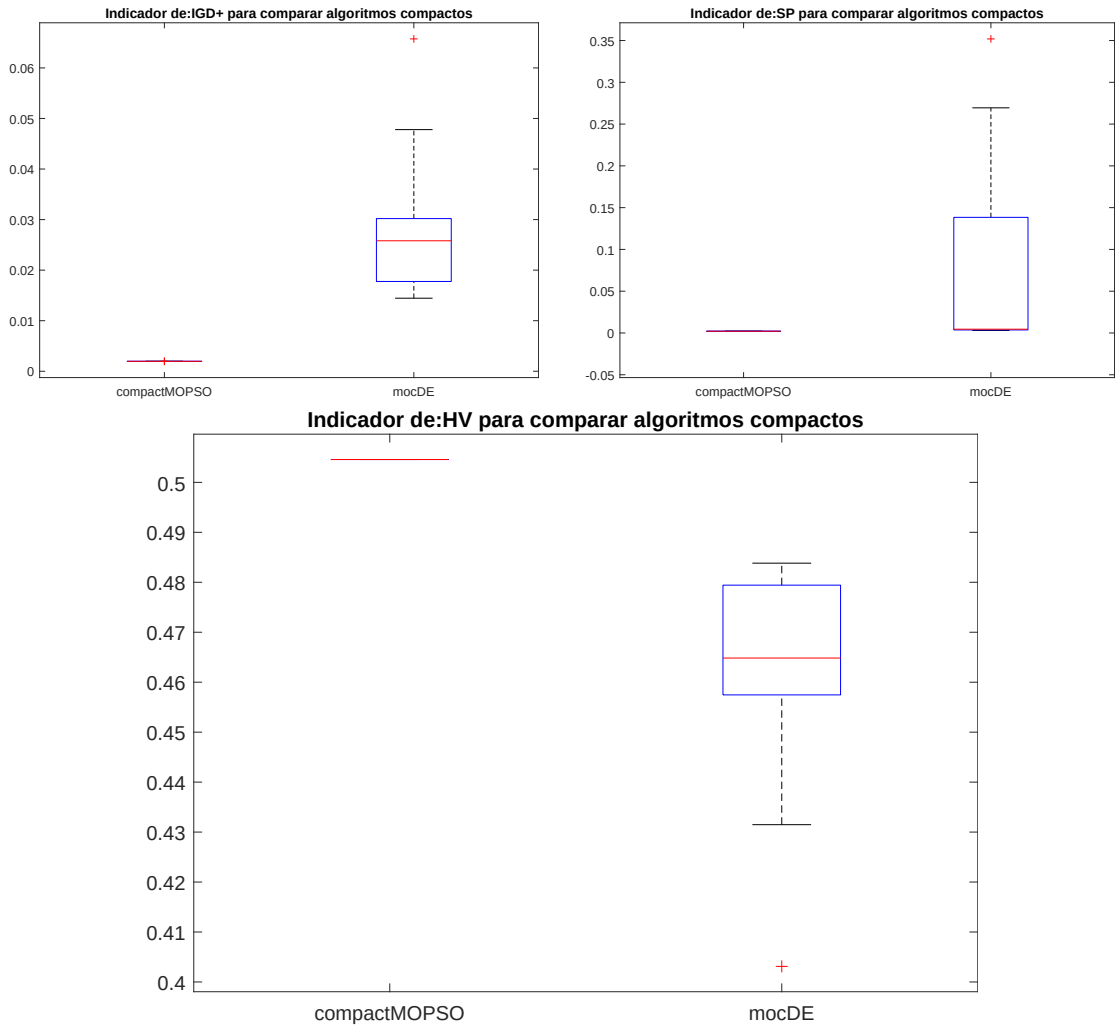


Figura C-44: Gráficas de caja de los tres indicadores utilizados para el problema de prueba ZDT6

Bibliografía

[1] Chang Wook Ahn. *Advances in Evolutionary Algorithms: Theory, Design and Practice (Studies in Computational Intelligence)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Chang Wook Ahn and R.S. Ramakrishna. Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4):367–385, August 2003.

[3] Abu Syed Md. Masud Ching-Lai Hwang. *Multiple Objective Decision Making, Methods and Applications: A State-of-the-Art Survey*. Lecture Notes in Economics and Mathematical Systems 164. Springer-Verlag Berlin Heidelberg, 1 edition, 1979.

[4] Maurice Clerc and James Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):57–73, February 2002.

[5] W.J. Cody. Rational chebyshev approximations for the error function. *Mathematics of computation AMS*, 23(107):631–637, 1969.

[6] Carlos A. Coello Coello. *Introduccion a la Computación Evolutiva (Notas del curso)*. `http://delta.cs.cinvestav.mx/~ccoello/compevol/apuntes.pdf`.

[7] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1):63–69, 1997.

[9] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, Feb 2011.

[10] Kenneth Alan De Jong. On the organization of intellect, 1964.

[11] Kenneth Alan De Jong. An analysis of the behavior of a class of genetic adaptive systems., 1975.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr 2002.

[13] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 825–830, May 2002.

[14] Kalyanmoy Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evol. Comput.*, 7(3):205–230, September 1999.

[15] Kalyanmoy Deb and Deb Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[16] Juan J. Durillo, José García-Nieto, Antonio J. Nebro, Carlos A. Coello Coello, Francisco Luna, and Enrique Alba. *Multi-Objective Particle Swarm Optimizers: An Experimental Comparison*, pages 495–509. Springer Berlin Heidelberg, 2009.

[17] Russell C. Eberhart and Yuhui Shi. *Comparison between genetic algorithms and particle swarm optimization*, pages 611–616. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[18] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003.

[19] David Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, USA, 1995.

[20] Lawrence J. Fogel. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[21] C. M. Fonseca, L. Paquete, and M. Lopez-Ibanez. An improved dimension-sweep algorithm for the hypervolume indicator. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1157–1163, July 2006.

[22] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[23] David E. Goldberg, Kalyanmoy Deb, and James H. Clark. Genetic algorithms, noise, and the sizing of populations. *COMPLEX SYSTEMS*, 6:333–362, 1991.

[24] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, Sept 1999.

[25] Georges R. Harik, Fernando G. Lobo, and David E. Goldberg. The compact genetic algorithm. *IEEE Transactions On Evolutionary Computation*, 3(4):287–297, November 1999.

[26] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.

[27] Hisao Ishibuchi, Hiroyuki Masuda, and Yusuke Nojima. A study on performance evaluation ability of a modified inverted generational distance indicator. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 695–702, New York, NY, USA, 2015. ACM.

[28] Kenneth A. De Jong. *Evolutionary Computation, A Unified Approach.* The MIT Press, 2006.

[29] J. Kennedy. Bare bones particle swarms. In *Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE*, pages 80–87, April 2003.

[30] J Kennedy and R Eberhart. Particle swarm optimization. In *1995 IEEE International Conference on Neural Networks Proceedings, vols 1-6*, pages 1942–1948, 345 E 47TH ST, NEW YORK, NY 10017 USA, 1995. IEEE.

[31] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1671–1676, 2002.

[32] James Kennedy, Russel C. Eberhart, and Yuhui Shi. *Swarm intelligence.* Morgan Kaufmann Publishers, San Francisco, 1st edition edition, 2001.

[33] X. Li. Niching without niching parameters: Particle swarm optimization using a ring topology. *IEEE Transactions on Evolutionary Computation*, 14(1):150–169, Feb 2010.

[34] Adriana Lara López. Using gradient based information to build hybrid multiobjective evolutionary algorithms, 2012.

[35] K. Miettinen. *Nonlinear multiobjective optimization.* Kluwer Academic Publishers, Boston, 1999.

[36] E. Mininno, F. Neri, F. Cupertino, and D. Naso. Compact differential evolution. *IEEE Transactions on Evolutionary Computation*, 15(1):32–54, Feb 2011.

[37] Ernesto Mininno, Francesco Cupertino, and David Naso. Real-valued compact genetic algorithms for embedded microcontroller optimization. *IEEE Transactions on Evolutionary Computation*, 12(2):203–219, April 2008.

[38] Antonio J. Nebro, Juan J. Durillo, José García-Nieto, Carlos A. Coello-Coello, Francisco Luna, and Enrique Alba. Smpso: A new pso-based metaheuristic for multiobjective optimization. In *2009 IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making*, Sheraton Music City Hotel, Nashville, TN, USA, March 30 - April 2 2009.

[39] Ferrante Neri, Ernesto Mininno, and Giovanni lacca. Compact particle swarm optimization. *Information Sciences*, 239:96–121, February 2013.

[40] E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. In *Intelligent Engineering Systems Through Artificial Neural Networks (Proc. ANNIE'98)*, volume 8, pages 253–258, 1998.

[41] M.E.H. Pedersen. Good parameters for particle swarm optimization. *Hvass Lab.*, 23, 2010.

[42] Wei Peng and Q. Zhang. A decomposition-based multi-objective particle swarm optimization algorithm for continuous optimization problems. In *2008 IEEE International Conference on Granular Computing*, pages 534–537, Aug 2008.

[43] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[44] A. Prügel-Bennett. Benefits of a population: Five mechanisms that advantage population-based algorithms. *IEEE Transactions on Evolutionary Computation*, 14(4):500–517, Aug 2010.

[45] Jason R. Schott. Fault tolerant design using single and multicriteria genetic algorithm optimization, May 1995.

[46] Ingo Rechenberg. Evolutionsstrategie, optimierung technischer systeme nach prinzipien der biologischen evolution. *Feddes Repertorium*, 86(5):337–337, 1975.

[47] Amaresh Sahu, Sushanta Kumar Panigrahi, and Sabyasachi Pattnaik. Fast convergence particle swarm optimization for functions optimization. *Procedia Technology*, 4:319 – 324, 2012.

[48] Yuntao Shi and Russell Eberhart. A modified particle swarm optimizer. In *The 1998 IEEE International Conference on Evolutionary Computation Proccedings*, William A. Egan Civic and Convention Center, Anchorage, Alaska, May 4-9 1998.

[49] Yuntao Shi and Russell Eberhart. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, La Jolla Marriott Hotel, La Jolla, California, USA, July 16-19 2000.

[50] Margarita Reyes Sierra and Carlos A. Coello Coello. Improving pso-based multi-objective optimization using crowding, mutation and in-dominance. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*, EMO'05, pages 505–519. Springer-Verlag, 2005.

[51] Margarita Reyes Sierra and Carlos A. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *INTERNATIONAL JOURNAL OF COMPUTATIONAL INTELLIGENCE RESEARCH*, 2(3):287–308, 2006.

[52] Rainer Storn and Kenneth Price. Differential evolution, a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, 1997.

[53] Jesus Moises Osorio Velazquez, Carlos A. Coello Coello, and Alfredo Arias-Montano. Multi-objective compact differential evolution. In *2014 IEEE Symposium Series on Computational Intelligence*, Orlando, Florida, USA, December 9-12 2014.

[54] Saúl Zapotecas Martínez and Carlos A. Coello Coello. A multi-objective particle swarm optimizer based on decomposition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 69–76. ACM, 2011.

[55] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, Dec 2007.

[56] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.*, 8(2):173–195, June 2000.

[57] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Multiobjective optimization. chapter Quality Assessment of Pareto Set Approximations, pages 373–404. Springer-Verlag, Berlin, Heidelberg, 2008.

[58] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, PPSN V, pages 292–304, London, UK, UK, 1998. Springer-Verlag.